Optical Tracking Control of a Differentially-Driven Wheeled Robot

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Loren Yeo

B. Eng. (Hons.)

School of Aerospace, Mechanical and Manufacturing Engineering

College of Science, Engineering and Health

RMIT University

Bundoora, Victoria

December 2012

To my mother

and

in memory of my father

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Loren Yeo

1 December 2012

Abstract

Mobile robotics has become an increasingly ubiquitous technology in modern times. A typical example is the wheeled mobile robot (WMR). In order for a WMR to function effectively, it must demonstrate excellent tracking control and localisation capabilities. This is achieved by having accurate and responsive control algorithms as well as high-precision sensor systems. However, this often requires complicated algorithms and expensive equipment.

This thesis proposes a system to show that good tracking performance can be achieved with moderately simple control algorithm and relatively inexpensive hardware. The platform used in this research was a differentially-driven wheeled robot constructed using the Lego Mindstorms NXT system. Positional tracking was provided by two Avago optical laser sensors commonly found on the computer mouse. The main programming environments were MATLAB and Simulink, along with several other open-source applications.

In the first part of the thesis, a PID-based system is presented along with the two control schemes. The first is a purely kinematic model and the second includes dynamic constraints. For both versions, a cascaded PID design was employed and settings were manually tuned. The final mathematical models were computationally simulated and their respective results were analysed and compared. Hardware validation was not conducted for this phase of the research as the simulation results suggested that the PID-based system may not produce the desired level of tracking performance.

The second part of the thesis explores a model reference adaptive control system. Lyapunov's direct method was used to achieve stability and convergence in the system. In contrast to the PID-based model, a vastly more accurate geometric localisation technique was applied. The research identified a number of shortcomings in current geometric localisation methods and suggested ways to mitigate them. In addition, a novel approach for detecting faulty sensor readings was introduced in conjunction with the development of a semi-redundant system.

The eventual theoretical model was tested using computer simulation, and the outcome was contrasted with the results of the PID-based system. This was followed by the construction of a prototype in order to verify the validity of the proposed model. Various configurations of the adaptive model were tested and compared: the two localisation methods, use of single and dual sensors, and application of semi-redundancy.

The thesis concludes with the analysis of results of the prototype testing. The theoretical propositions in the thesis were shown to be amply validated. Suggestions for future research work are also presented.

Acknowledgements

A research endeavour is akin to a long and challenging journey. Without the assistance, motivation and support provided by others along the way, the task would be immensely harder, if not impossible.

To primary supervisor, Prof Sabu John, I would like to express my deepest gratitude for his invaluable insight, guidance and support throughout the course of my research. I would also like to extend my sincere thanks to my secondary supervisor, Prof John Mo, for the input and help that he has provided me.

Special acknowledgement goes to Prof James Peyton Jones of Villanova University for his technical assistance with the Villanova University LEGO Real Time Target (VU-LRT) software.

Last but not least, I would like to offer my heartfelt thanks to my mother and family for all the love, encouragement, support and patience that they have shown me over the years. I would be nothing without them.

Table of Contents

List of Figures

List of Tables

Nomenclature

A	Location of first (rear) sensor
a	Distance between rear sensor and wheel baseline
a_{nb}, a_{nl}, a_{nr}	Normal acceleration at centre of wheel baseline, left or right wheel
a_{max}	Maximum allowable acceleration without slippage
a_{tl}, a_{tr}	Tangential acceleration of left or right wheel
a_x	Longitudinal acceleration of vehicle
a_y	Lateral acceleration of vehicle
B	Centre of wheel baseline
b	Distance between centre of gravity and wheel baseline
B_m, B_L	Viscous-friction coefficient of motor or load axle
C	Point of contact of the front wheel with the ground
C	Matrix containing Coriolis and centrifugal terms
c	Distance between front wheel and wheel baseline
D	Coefficient matrix
d	Track width of vehicle
E_a	Applied voltage
E_b	Back-EMF

e_{crit}	Base-2 exponent that corresponds to the critical radius
e_t	Tracking error
F_{max}	Maximum allowable force without slippage
F_x	Longitudinal force exerted by driving wheels
$F_{x'l}$, $F_{x'r}$	Longitudinal forces exerted by left or right driving wheel
F_y	Lateral force exerted by driving wheels
$F_{y'l}$, $F_{y'r}$	Lateral forces exerted by left or right driving wheel
G	Centre of gravity of vehicle
h_q	New input for linearised system
i_a	Armature current
I_z	Moment of inertia of vehicle
J_m, J_L	Inertia of motor's rotor or load axle
K	Location of second (front) sensor
k	Distance between front sensor and wheel baseline
K_b	Back-EMF constant
K_r	Regressor matrix
K_t	Torque constant
L	Distance between the two sensors
L	Point of contact of the left wheel with ground
L_a	Armature inductance
M	Inertial matrix
m	Mass of vehicle
N	Normal force on vehicle
n_g	Gear ratio
P	Symmetric positive definite constant matrix
Q	Strictly positive and symmetric matrix
q	Tracking vector
$\tilde{q}$	Difference between current and desired values of tracking vector
q_d	Desired values of tracking vector
R	Point of contact of the right wheel of vehicle with the ground
r	Wheel (including tyre) radius
R_a	Armature resistance
r_{crit}	Critical radius at which computing precision exceeds sensor resolution
$r_{path,\ l}$, $r_{path,\ r}$	Path radii for left or right wheel
r_s	Radius of circular path formed by the centre of the wheel baseline
RMS	Root mean square
RMSE	Root mean square error
s_l, s_r	Distance covered by left or right wheel
s_1, s_2	Arc length formed by Sensor 1 or 2
SQUAL	Surface quality

T_m, T_L	Motor or load torque
$T_{L,l}$, $T_{L,r}$	Load torque on left or right driving wheel
u	Longitudinal velocity of vehicle
$\dot{u}$	Longitudinal acceleration of vehicle
U_i	Input vector for dynamic system
ULP	Unit in the last place
V	Lyapunov function candidate
v	Lateral velocity of vehicle
$\dot{v}$	Lateral acceleration of vehicle
x	Position of vehicle on X-axis
x'	Position of vehicle on local X-axis
$\dot{x}$	Velocity of vehicle on X-axis
x_a, x_b, x_l, x_r	X-coordinate of rear sensor, centre of wheel baseline, left or right wheel
x_p	X-coordinate of pivot wheel
Δx_G	Displacement of centre of gravity with respect to X-axis
Δx_1, Δx_2, Δx_A, Δx_K	Displacement detected by sensor on X-axis
$\Delta x_1'$, $\Delta x_2'$	Displacement detected by sensor on local X-axis
y	Position of vehicle on Y-axis
y'	Position of vehicle on local Y-axis
$\dot{y}$	Velocity of vehicle on Y-axis
y_a, y_b, y_l, y_r	Y-coordinate of rear sensor, centre of wheel baseline, left or right wheel
y_p	Y-coordinate of pivot wheel
Δy_G	Displacement of centre of gravity with respect to Y-axis
Δy_1, Δy_2, Δy_A, Δy_K	Displacement detected by sensor on Y-axis
$\Delta y_1'$, $\Delta y_2'$	Displacement detected by sensor on local Y-axis
α	Yaw (turning) acceleration of vehicle
α_l, α_r	Angular acceleration of left or right wheel
β	Angle between local X-axis and tangent to the turning arc
Γ	Symmetric positive definite constant matrix
γ	Angle between turning radii
ε_x, ε_y	Tracking error on X- or Y-axis
θ_m, θ_L	Angular displacement of motor or load axle
θ_p	Parameter vector
$\dot{\theta}_m, \dot{\theta}_L$	Angular velocity of motor or load axle
$\ddot{\theta}_m, \ddot{\theta}_L$	Angular acceleration of motor or load axle
λ	Strictly positive number
μ	Coefficient of friction of tyres
σ	Standard deviation
ψ	Yaw (turning) angle of vehicle

ω	Yaw (turning) velocity of vehicle
$\dot{\omega}$	Yaw (turning) acceleration of vehicle
ω_l, ω_r,	Angular velocity of left or right wheel
$\dot{\omega}_l$, $\dot{\omega}_r$	Angular acceleration of left or right wheel

Chapter 1: General Introduction

1.1 Introduction

The importance of robotic devices is reflected in their ever widening use. In mobile robotics, wheeled types are most commonly seen. Of the many varieties of wheeled mobile robots (WMRs), one of the popular configurations is that of the differential-drive system. This sort of robot is favoured for its simplicity in design and also for its manoeuvrability.

Key to the operation of an autonomous WMR is localisation – the ability to locate itself within a frame of reference. An example of a device that is able to track its position within a defined space is the ubiquitous computer mouse. The principle behind its optical tracking system can easily be applied to autonomous WMRs. Since this tracking system has a straightforward design that requires minimal computational power, it would be ideal for use on a simple and inexpensive WMR. Indeed, such a set-up has been tested on small-scale platforms (O'Hara and Kay 2003; Singh and Waldron 2004).

Apart from a capable tracking system, a WMR also requires an effective system to control and drive it. Underpinning this control algorithm is a mathematical model of the behaviour and properties of the WMR. The two main ways in defining a mathematical model are kinematic and dynamic modelling (Saha and Angeles 1989; Campion et al. 1996). Due to better practical realism, dynamic modelling has been increasing favoured over the kinematic approach.

1.2 Background

The potentially unlimited possibilities for mobile robotic applications have kept researchers excited for many years. While traditionally, robots have seen their biggest deployment in the manufacturing sector, they are gradually making inroads into other areas such as transportation, medicine and even consumer products. The advent of mobile robotics has opened up new fields of applications where conventional fixed-base robots would have been impractical. From automated guided vehicles in industrial settings to nascent self-driving cars, the field continues to grow rapidly. Furthermore, mobile robots are especially useful in hazardous applications such as planetary, deep-sea and mining explorations, as well as bomb disposal, landmine detection and clearance, etc.

A crucial factor that affects the performance of a WMR is its ability to keep track of its current location. To accomplish this, a wide range of methods have been employed. One of the oldest

and most rudimentary approaches is based on the idea of relative positioning and is called dead-reckoning. This is usually accomplished with the use of wheel encoders, but inertial measurement units (IMUs) such as accelerometers and gyroscopes have seen increasing rates of adoption in recent years. Another popular technique is depth sensing. This usually involves range finders that emit laser, infrared or sonar. In recent years, due to the rapid advancement in computer processing power, vision-based positioning using single or dual cameras systems have also become increasingly common. Also in use are triangulation, trilateration and multilateration techniques via the use of signal beacons or transponders. The fixed reference points provide instantaneous location on a grid and are based on the principle of absolute positioning. A large-scale example of this would be the Global Positioning System (GPS). However, it has inadequate accuracy as well as limited use within buildings. Additional sensors may not necessarily improve robot performance. Indeed, they may slow processing time, increase heft and raise energy consumption.

A widely-known example of a tracking system is found on the computer mouse. The optical laser sensors installed in the latest computer mice work on the principle of optical flow, and are cheap, highly accurate and require very minimal computational power to operate. Also, they are not susceptible to errors caused by tyre slippage or discrepancy in wheel sizes. This optical sensor technology has existed for quite some time now, but there has not been any commercial or large-scale deployment of such it for navigation or tracking purposes. In one of the first dead-reckoning systems that introduced the use of optical mouse sensor to replace wheel encoders, an analogue compass was also employed to provide yaw readings (Silva et al. 2002). Unfortunately, the relatively low precision of the compass affected the overall accuracy of the system. Later proposals (Bonarini et al. 2004; Bonarini et al. 2005) employed the use of two optical sensors that fully separated the robot's localisation from its kinematics. Despite the high level of precision offered by the sensors, the final outcome was still appreciably affected by systematic and non-systematic errors. Further studies (Minoni and Signorini 2006; Palacin et al. 2006) looked into the factors that impact upon the accuracy of optical mouse sensors in order to understand why they have not been able to deliver the accuracy expected of them.

In the following years, there have been continuing attempts to improve the tracking accuracy of robotic vehicle that use mouse sensors for navigation. Naturally, there is also interest in extending the application of mouse sensors to a wider range of conditions - such as the outdoors (Jackson et al. 2007). These sensors have been suggested for use in robot swarms (Gustafson et al. 2005) as well as unmanned flying vehicles (Thakoor et al. 2004). Some have added other types of sensors or devices to work alongside mouse sensors in order to supplement or enhance their performance. A number of these sensor-fusion platforms have added inertial measurement units to provide better versatility especially when the mouse sensors are deployed in conditions where the tracking surface is not uniformly even (Hyun et

al. 2009; Park et al. 2009; Dille et al. 2010). Other systems employ additional cameras that are not attached to the vehicles but placed at certain locations (Sekimori and Miyazaki 2005; 2007). However, the practical limitations of such systems are obvious.

Apart from the precision of the sensors, the effectiveness of a WMR's tracking control system also depends very much on the soundness of its underlying theoretical model. In recent years, dynamic-based modelling (Zhao and BeMent 1992; Zhang et al. 1998; 2003) has become increasingly popular as techniques and technology continue to improve. The kinematic approach (Oriolo et al. 2002) is well-established and retains much popularity due to its computational simplicity. However, by not accounting for dynamic forces, this method faces severe practical limitations even at relatively low speeds (Boyden and Velinsky 1994a; Hong et al. 1999). As a result, under- or over-steer often occurs due to significant wheel/tyre slippage. While it is very difficult to take every dynamic effect into account, dynamic modelling is generally a more realistic solution than a purely kinematic approach.

1.3 Motivation, Objectives and Research Questions

The issue of tracking control precision is highly important in the study of autonomous robots. Without reliable solutions, applications will remain limited. Over the years, accuracy in tracking control has improved markedly. Numerous methodologies and technologies have been introduced and tested. However, costs, complexity and required processing power have escalated with increased precision. At present, tracking sensor systems used on WMR platforms range from light and cheap encoders to ultrasonic sensors as well as bulkier and more expensive laser scanners and camera set-ups. The challenge is to produce an accurate tracking system without the exorbitant costs normally associated with high-precision platforms.

A prospective alternative is the optical mouse sensor. Based on its extremely high scale of precision, albeit with known limitations, it was decided this could be a viable option to investigate. Currently-available mouse sensors can detect tiny movements ranging from 1.27×10^{-2} mm (2000 dpi) (Avago 2009b) to 4.48×10^{-3} mm (5670 dpi) (Avago 2011), and are more tolerant than previous models of slight height changes between sensor and surface. Yet, despite such high level of sensor precision, studies have repeatedly shown that WMRs using such sensors have a tendency to deviate by a margin of tens to hundreds of millimetres from their real-world position after travelling a distance of only a few metres. It remains to be thoroughly explained why the magnitude of the tracking error grows so quickly.

It is clear that such rates of error accumulation place serious limitations on the applicability of this particular type of technology for navigational purposes. Thus, it is important to find out

why mobile robots are unable to deliver the level of accuracy commensurate with their high-precision optical mouse sensor systems.

Ultimately, it is the aim of this research to explore the feasibility of developing an accurate tracking system that is inexpensive, has low processing demands, requires no assistance from external reference devices, and is easily deployable. To achieve this goal, it is vital to investigate why others have not achieved the optimal results as expected when using such high-precision optical sensors. Also, various tracking control methods would be analysed and compared, and the most effective one adopted for use.

The research would be guided by the following questions:

1. Is the tracking control algorithm robust enough for its purpose?
2. How does the performance of kinematic and dynamic models compare?
3. Could relatively poor tracking accuracy be attributed to inherent sensor limitations?
4. How accurate are the odometric calculations for localisation?
5. Are there any other yet unknown factors that have an effect on tracking performance?
6. What could be done to improve the performance and viability of systems that use this technology?

It is hoped that the final research outcome would be able to answer these questions and deliver an accurate tracking control platform without the high cost or complexity usually associated with high-precision systems. This would be demonstrated by using off-the-shelf components to build an inexpensive vehicular platform coupled with a pair of optical mouse sensors to provide feedback information with a level of redundancy.

1.4 Thesis Outline

This chapter introduces the main idea of the thesis and outlines the motivation and objectives of the research. The rest of the thesis is organised in the following manner:

Chapter 2 provides a literature review of all the salient topics that are discussed in this thesis. This includes kinematic and dynamic modelling techniques, navigation and tracking methods, common sensor technologies and methodologies, and various types of control systems.

Chapter 3 explores in detail the theoretical model of the wheeled robot. Kinematic and dynamic models are analysed mathematically, and friction modelling is introduced.

Chapter 4 describes the computational design of a PID-based WMR. All components and algorithms are modelled in software.

Chapter 5 presents the simulation results of the PID model. The outcome directly affects the decision on whether to proceed to hardware validation or adopt a new or revised mathematical model.

Chapter 6 examines the mathematical model of an adaptive system in detail. Two techniques for calculating odometry are explored and a partially-redundant system is proposed.

Chapter 7 explains the computational design of an adaptive system. All components and algorithms are modelled in software.

Chapter 8 illustrates the construction of the test prototype. All hardware components and their usage are described in detail. The software used for the research is also outlined.

Chapter 9 presents both the simulation and validation results of the adaptive model. Single- and dual-sensor configurations are compared, and the partially-redundant system is put to the test.

Chapter 10 consists of the overall conclusions of the research and proposes additional work for future consideration.

1.5 List of Publications

Yeo, L., S. John, et al. (2008), Simulation of an Optical Tracking Control System of a Differentially-Driven Wheeled Mobile Robot, Proceedings of the International Conference on Modeling, Simulation & Visualization Methods (MSV 2008), pp. 41-47.

Yeo, L., S. John, et al. (2009), Simulation of an Optical Tracking Control System of a Differentially-Driven Wheeled Robot, International Journal of Computer Aided Engineering and Technology, vol. 2(1): pp 15-29.

Chapter 2: Literature Review

Every research project begins with a comprehensive review of the accumulated knowledge within the field. Only with a thorough understanding of the subject can research then be undertaken. The literature review will include topics such as mathematical modelling methods as well as sensor technologies and methodologies.

2.1 Mathematical Modelling Methods

The most logical method for the modelling of properties and behaviour of WMRs is mathematically-based. The process usually begins with the derivation of kinematic equations to describe the movement of the WMR. This is followed by the formulation of dynamic equations that take forces into account. Naturally, the latter approach provides a more realistic model.

Currently, many modelling methods for differential drive WMRs ignore the presence of longitudinal or lateral wheel/tyre slippage in order for path planning to be based purely on kinematics. Even with the smoothest of trajectories, this model is highly idealistic and prone to inaccuracies in practical applications. More recent proposals have included dynamic considerations to minimise longitudinal slip due to excess torque, as well as lateral slip due to turning manoeuvres. There has also been suggestion on the use of a Gauss-Newton algorithm to predict positional errors (Seyr and Jakubek 2005). It remains to be seen how well predictive methods perform when unforeseen situations arise.

2.1.1 Kinematic Modelling

The most basic mathematical technique for describing the movement of an autonomous robot is based on kinematics. Hence, some form of kinematic modelling is employed in the motion control of most mobile robots. With knowledge of wheel diameter, track width, and other salient vehicular dimensions a tracking-sensor information, the moving velocity and acceleration of the WMR can be derived from the angular velocity of its wheels and vice versa. Additionally, the rotational displacement and turning velocity of the WMR can be obtained by analysing the difference between the angular velocities of its wheels and vice versa.

For the mobile robot to get to a destination, a route is first defined and then divided into numerous checkpoints. The allotted time for the WMR to get from one checkpoint to the next, as well as the locational information of the checkpoint relative to neighbouring ones will

determine the required travelling speed and direction for the particular segment of the route. This is essentially how the motion of a typical WMR is kinematically controlled.

By ignoring dynamic forces, a purely kinematic approach has been shown to produce unrealistic results at elevated loads and speeds (Boyden and Velinsky 1994a; Hong et al. 1999). As a consequence, there are significant limitations to the applicability of solely kinematic systems (Boyden and Velinsky 1994b). Hence, in recent years there has been a trend towards the application of dynamic modelling to mobile robotics.

2.1.2 Dynamic Modelling

Currently, there are several mathematical approaches to dynamic modelling of WMRs. Newton-Euler (Zhang et al. 1998) and Euler-Lagrange (d'Andrea-Novel et al. 1991) formulations of mechanics are both well-established and are naturally most commonly adopted. However, a more recent method proposed by Kane and Levinson (1983) has also been well-received by researchers (Thanjavur and Rajagopalan 1997; Nukulwuthiopas et al. 2002).

Essentially, all three approaches are simply variants of classical mechanics. Lagrangian mechanics is a re-formulation of Newtonian mechanics while Kane's method is an adaptation of Lagrangian mechanics. Whereas the Newtonian approach concentrates on the balance of forces, the Lagrangian technique focuses on the conservation of energy. As for Kane's method, it was developed primarily for analysing non-holonomic systems such as multi-body objects.

Considering that the WMR used in this study is of a relatively simple design, a straightforward analysis was initially achieved by adopting the Newtonian approach. The Euler-Lagrange method is used in a later model incorporating adaptive control.

2.2 Navigation and Tracking

The most fundamental requirement for the successful navigation of a mobile robot is its ability to know its position with an acceptable degree of certainty at any given time (Borenstein et al. 1997). This concept is called localisation.

One of the methods that has been used for a long time is that of dead-reckoning. It is popular because of the simplicity of the technique. Essentially, the current position of an object of interest is determined by how far it has moved in all pertinent axes from its last known

location. This can be derived from information such as distance, speed, heading and length of time between measurements. Without a priori knowledge of a previous reference point, this approach will not work. A major drawback of this method is that inaccuracies in the measuring these variables would result in positional errors that accumulate over time.

There are two kinds of errors inherent in dead-reckoning: systematic and non-systematic. The first type is directly related to the properties and characteristics of the vehicle. Factors such as unequal wheel sizes, inaccurate measurements of wheel diameter, track, or wheel base, would result in the erroneous calculation of a vehicle's position. These biased errors can be reduced by careful calibration of the system. The second type of error is often harder to identify and minimise. Non-systematic errors are usually random and thus impossible to predict. Examples of such errors are sensor noise, environmental fluctuations, etc.

When a mobile robot is able to localise itself, the next step would be to develop a tracking algorithm where it is able to follow a prescribed route or head towards a particular destination. The type of control strategy employed is determined by the objective of the operation. The methods most commonly seen are path following, trajectory tracking and point-to-point stabilisation.

If it is necessary for the WMR to adhere to a desired path, then a route within the reference frame must be mapped out beforehand. This type of control strategy is called path following. In this case, the WMR is allowed to set its own speed as long as it stays on the path.

However, if the WMR were to be required to not only stick to a given course but also be at particular locations along the course at specific times, then its speed will be regulated by time constraints. This form of control strategy is termed trajectory tracking.

In the case where the WMR is only required to travel to a target location without a pre-defined route or time limit, it is essentially free to decide its own path to the destination as well as its speed along the way. This mode of control strategy is called point-to-point stabilisation.

An important caveat regarding the latter approach is that the WMR must be able to detect and avoid obstacles and hazards while on its way to its final destination. This would require additional sensors and complex algorithms which would lead to increased computational demands. If there are no maps or guides available from the outset and the WMR is required to chart its own maps, then this is the only logical option. In contrast, if a clear and safe route can be specified from the beginning, as required by the first two strategies, then such problems may be avoided. Depending on the operational requirements, each of these schemes has its advantages and disadvantages (Sarkar et al. 1993, 1994).

As no system is perfect, it is inevitable that tracking errors will occur at some point. During the map-building phase, tracking errors can lead to serious distortions and would require numerous passes and clever algorithms to arrive at some sort of convergence. If the reference maps are poorly charted due to tracking errors, it would be impossible for the WMR to later follow its intended path. Instead, the WMR would attempt to adhere to a distorted map and end up somewhere it is not supposed to be. At present, a common approach to localisation and map-building is the employment of probabilistic means (Leonard and Durrant-Whyte 1991; Durrant-Whyte and Bailey 2006) to match and merge visually-detected features. Using this method, a WMR is able to determine its own location and at the same time build up a map as it goes along. This technique is called Simultaneous Localisation and Mapping (SLAM) (Montemerlo et al. 2002, 2003).

The situation with tracking error is less critical if an acceptably accurate map has already been pre-programmed in a robot's memory. The map could have been input manually or generated autonomously from previous surveys. Current image analysing technology is quite capable in aiding the localisation and orientation of a robot using comparisons of visual features, such as landmarks or contours, with its stored maps. Similarly, sonar systems are able to discern notable features by examining signal echoes. Another popular method is the measurement of distance travelled by each wheel via the use of wheel encoders. These image, sonic or odometric feedback systems help to keep the robot on a desired path.

Away from simulated settings, it remains to be seen how well current methods cope with challenging real-life situations such as environments with few noticeable features, or surfaces that are severely uneven or slippery. A simple real-world test for the tracking accuracy of a WMR is the calibration procedure called the UMBmark (Borenstein and Feng 1995). It would one of the tests included in this study although the calibration process of UMBmark would not be used.

2.3 Common Mobile Robot Sensor Technologies and Methodologies

The navigational ability of an autonomous WMR depends critically on how well it senses its surroundings. Not only must the robot be able to "see" where it is going, it must also be able to "know" its current location within a specified frame of reference. Without some form of sensor, the WMR will simply not be able to find its way around.

Before a decision is made in regards to the type of navigational sensor to be adopted, an examination of common sensor technologies and methodologies must first be conducted. The inherent advantages and disadvantages of these technologies will be compared with one another in order to make the most appropriate selection to meet the objectives of this study.

2.3.1 Dead Reckoning Sensors

In the art of navigation, one of the oldest ways of determining the current location of a traveller or vessel is a method called dead reckoning. Essentially, with a known previous position and a well-documented course leading from that position to the current one, the present location can be deduced from the distance between the two points. If the distance travelled is not known directly, that information can often be derived alternatively from the speed and time of travel. This technique is straightforward and well-tested. Hence, it is widely-adopted for navigational purposes including mobile robotic navigation.

A basic form of dead reckoning is a method called odometry. The word is derived from odometer - the American term for a device that measures vehicular displacement, especially that of a motor car. This is a direct approach where travelled distance is determined solely by using instrumentation aboard the object of interest. In contrast, indirect methods, such as triangulation or multilateration, obtain displacement information via derivation.

2.3.1.1 Optical Encoders

Mobile robots come in many forms. Some are airborne while others may have legs. However, the most common types employ wheels or continuous tracks for locomotive purposes. Most of these wheeled robots use rotary encoders to measure travelling speed and distance. Rotary encoders are generally used for measuring the angular velocities of rotating shafts and axles. Knowing the angular velocity as well as the overall diameter of a particular wheel and its tyre, the speed and distance travelled by any wheeled vehicle can be easily determined.

For wheeled robots, encoders are often used for monitoring the rotation of a pair of wheels, each located on opposite sides of the robot. When the robot is travelling in a straight line, the angular speed of equal-sized wheels on opposite sides are identical, save for equipment tolerances. If the robot is negotiating a turn, the rotational velocities of both wheels will naturally differ from each other. Using this information, the robot's direction can be obtained in addition to its moving speed. When the robot operates over a known period of time, the travel route as well as velocity, displacement and direction can all be determined quite easily.

Magnetic and optical encoders are the most common kinds of rotary encoders encountered nowadays. For mobile robotics, optical encoders are generally preferred. The operation of an optical encoder typical involves a focused ray of light interacting with a patterned disc affixed to a rotating shaft. The result is a pulsed signal which is then picked up by a photosensor. This signal is digitally analysed by a microprocessor to obtain information on angular velocity and position.

The purpose of the coded pattern is to produce a binary output from the interaction with the light ray. This outcome will depend on where the light lands on the disc. The design of the encoder will determine whether the emitter and photodetector are located on the same side or opposite sides of the patterned disc.

In the former case, the disc pattern is arranged in such a way where there are contrasting regions of high and low reflectivity. Consequently, the intensity of the light that bounces off the patterns can be sorted into two levels: high and low. This reflected signal is then picked up by the sensors and interpreted as a binary output.

In the latter case, a binary signal is generated by the selective obstruction of the light ray. In order for light to pass through intermittently, the disc is either transparent of perforated in certain regions specified by the coded pattern. The light ray is thus transformed into an on-off signal which can be detected by sensors on the other side of the wheel and then translated into binary data.

Optical encoders have two main varieties: incremental and absolute. For incremental encoders, angular velocity is measured directly while relative position is derived. In contrast, for absolute encoders, angular position is measured directly while velocity is derived. Due to the fact that incremental encoders contain fewer sensors, they are usually much cheaper than absolute types.

The main disadvantage of rotary encoders is that they are prone to odometric errors due to their inability to detect wheel slippage. An encoder will report the rotary speed of a wheel (or axle) that is being monitored regardless of whether there is full traction between the wheel (or tyre) and travelling surface. If there is slippage during vehicle acceleration, onboard encoders will likely report the vehicle speed (and distance travelled) to be higher than in reality. On the other hand, if slippage occurs during vehicle deceleration, the speed of the vehicle (and distance travelled) as determined by onboard encoders will tend to be lower than it actually is.

Odometric errors accumulate and are unbounded. Thus, even a small but consistent amount of slippage can quickly add up to level where it begins to have a substantial impact on operational accuracy. WMRs that rely on wheel encoders for positional tracking require quite some effort to calibrate (Borenstein and Feng 1995). Furthermore, it is essential for slippage to be minimised, so vehicle speeds are often kept relatively low.

2.3.1.2 Optical Mouse Sensors

The computer mouse was invented by Douglas Engelbart at the Stanford Research Institute (now called SRI International) in 1963. It was designed as an input device with two wheels perpendicular to each other and attached to potentiometers allowing it to track its movement along the horizontal and vertical axes. Engelbart's chief engineer, Bill English, who helped build the first mouse at SRI later moved to Xerox PARC and developed the ball mouse in 1972. Using a combination of optical encoders and mechanical rollers actuated by a ball, this design became the de-facto standard until the 1990s when it began to be supplanted by fully optical systems.

The optical mouse sensors seen today come in two types: light-emitting diode (LED) and laser. LED sensors tend to be older and are slowly being replaced by laser versions. While the source of illumination may differ in both variants, the principle behind how they work is identical.

The purpose of the LED or laser is to light up a small area of a surface to the extent where an image can be clearly captured by a tiny camera within the sensor. This process is repeated continuously as the mouse is moved across a surface. Using digital signal processing, the images are compared to one another to determine the distance and direction travelled. This principle is called optical flow (Beauchemin and Barron 1995).

The major advantage of laser over LED is light coherence. Specifically, coherent illumination allows images to be acquired with higher contrasts compared to a non-coherent light source like LED. The obvious benefits are better sensitivity and precision. This means that surfaces that have in the past proved to be tricky for LED sensors, such as low-detail or glossy planes, no longer pose a problem as shown in Fig. 2.1. Manufacturers have claimed that laser sensors can provide a contrast improvement of up to 20 times over LED counterparts (Agilent Technologies 2004; Teo 2006).

The ability of an optical mouse sensor in detecting movement is called its sensitivity. This is usually measured in counts (sometimes called dots) per inch, i.e. cpi or dpi. The typical sensitivity of sensors that are currently deployed range from 2000 dpi to 5670 dpi. This translates to a resolution that ranges from 1.27×10^{-2} mm to 4.48×10^{-3} mm respectively. This level of precision is virtually unmatched by most other types of sensors. Nowadays, some of the high performance sensors can reach a sensitivity of up to 8200 dpi (Avago 2012).

One of the drawbacks of optical mouse sensors is the tendency for reading errors to occur if there is just a bit of fluctuation in the gap between the sensor and tracking surface. Compared to previous models, newer sensors are generally more tolerant of such height variation. Refer

to Figs. 2.2 and 2.3 for a comparison of Avago sensors ADNS-2051 (released in 2005) and ADNS-6010 (released in 2006).

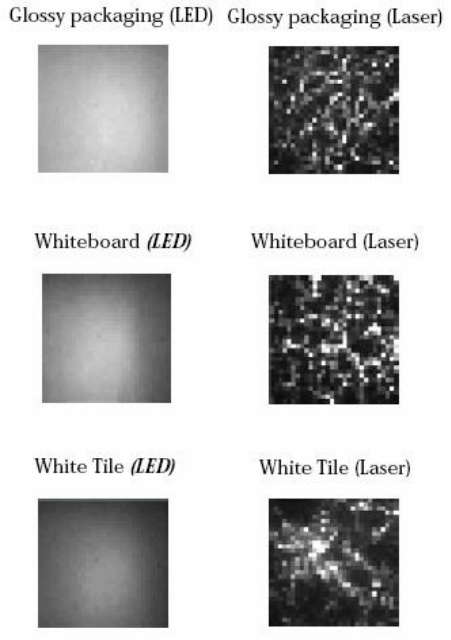


Fig. 2.1 Difference in image contrasts using illumination by laser and LED (Logitech 2004)

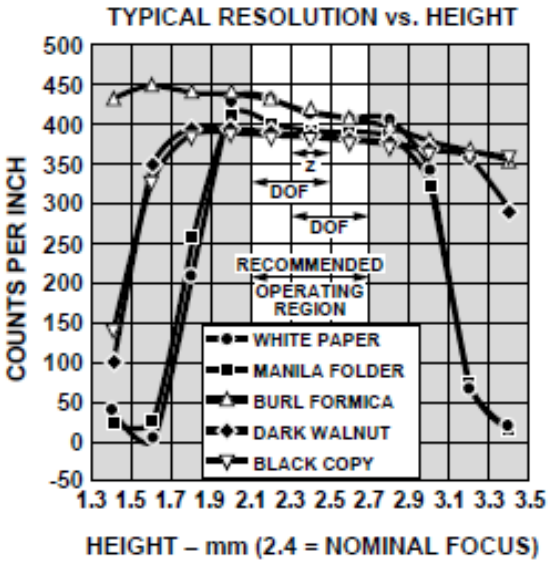


Fig. 2.2 Recommended sensor height for ADNS-2051 (Avago 2009a)

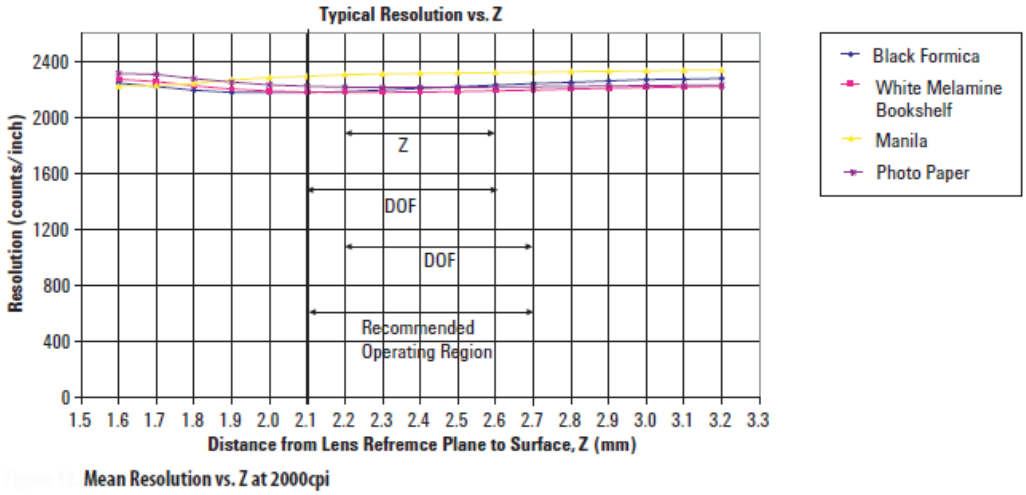


Fig. 2.3 Recommended sensor height for ADNS-6010 (Avago 2009b)

2.3.2 Range Sensors

Range (or depth) sensors have long been used in mobile robotics for obstacle avoidance, landmark detection, localisation and map building. The general principle behind the workings of these sensors is quite simple. An initial signal is emitted. This signal will travel until it makes contact with an obstacle and reflect back to the sensor. The total elapsed time is then processed and the distance between the sensor and obstacle can be determined. By analysing a series of emitted and reflected signals, approximate profiles of the obstacles within the sensor's view can be shaped. At the same time, the robot will be able to ascertain its location with respect to the surveyed environment. As the technique known as simultaneous localisation and mapping (SLAM) becomes increasingly popular, so has the adoption of depth sensors.

2.3.2.1 Laser Range Finders

In recent years, laser is increasingly used in range-finding applications. This is due to its high accuracy and fast response. Also, advancement in laser technology has greatly improved measuring distance to a range that was previously possible only with the use of radio or sound waves. In vacuum and air, the propagation capabilities of laser are excellent. Hence, the achievable range of measurement is very good. On the other hand, the transmission of high frequency waves, such as light, is severely inhibited by absorption effects of a denser medium like water. Therefore, underwater applications are limited to those that require a short operation range. It is important to note that laser light can fall within the visible range or within invisible spectrums such as infrared or ultraviolet.

Laser is essentially a form of highly-concentrated light. Hence, its speed of travel is identical to that of light and is similarly influenced by physical properties of the propagation medium, such as its composition, density, temperature, pressure and humidity. For light and other electromagnetic waves travelling through medium like air, the effects of variation in temperature, pressure and humidity are very small. When distances involved are short, they are essentially negligible. In contrast, the speed of sound in air is noticeably affected by temperature. For example, at atmospheric pressure, the speed of sound at 20°C is 343.4 m/s while that at 30°C is 349.2 m/s.

When the speed of a travelling wave is affected by changes in properties of the propagation medium, the phenomenon is called refraction. In pure vacuum, refraction does not occur and its refractive index is nominally set at the value of 1. The refractive indices of all other mediums are compared to this baseline. Using the modified Edlén equation (Edlén 1966), the refractive index of air is calculated to be 1.00027 at standard temperature and pressure and with a relative humidity of 50%. This figure is hardly different from that of a vacuum and shows the weak refractive effect of air. A change of 10°C in temperature results in a variation of less than 0.001% in the refractive index. A change of 30% in relative humidity results in a variation of about 0.0002% in the refractive index. Thus, it is quite evident that changes in properties of air have little effect on refraction. For short-range measurements in air, refraction can safely be ignored. For very long distances, refraction can distort the direction of a laser beam as well as contribute to a phenomenon called scintillation.

Scintillation occurs when pockets of atmospheric turbulence or thermal fluctuations lead to localised variations in the refractive index. A laser passing through such a region may undergo constructive and destructive interference. Ultimately, this can give rise to a change in signal intensity and phase. As indicated previously, the refractive property of air is very small. Thus, scintillation is only an issue when distances are large.

Aside from atmospheric disturbances, suspended particles in the air can also hinder the effective propagation of a laser beam by scattering it. Aerosols, such as dust and fog, can greatly affect the clarity of air and hence limit the operational distance of a laser range finder. Under such circumstances, the actual target surface may be obscured, thus leading to erroneous depth readings.

The capabilities of laser range finders are not limited by the characteristics of the transmission medium alone. Surface attributes of targeted objects are another significant factor. An almost perfectly-black surface will absorb most of the energy of an incident laser beam and result in hardly any detectable reflection. On the other hand, maximum reflectivity may not be desirable either. An incident laser beam arriving at a smooth and mirror-like surface will reflect in a narrowly-defined direction without losing much focus. This phenomenon is called

specular reflection. So, unless the sensor is in the path of the reflected beam, no signal would be recorded. Indeed, slight roughness on the surface of an object aids the detection of a laser on a targeted spot. This is because the diffuse reflection of a laser beam in multiple directions substantially increases the chances of detecting a reflected signal. Diffuse reflection is also known as Lambertian reflection. Another serious problem is that of transparent surfaces. Laser beams travel right through them with little or no reflection. If the medium is thick, the laser beam may also undergo considerable refraction. As such, readings would be seriously flawed.

The shape and external features of a targeted object and its environment can also affect the accuracy of a laser range finder. From certain angles, surfaces may be obscured. Furthermore, echoes caused by multiple deflections of a laser beam can produce completely misleading results. This problem is called cross-talk and is much less severe with laser than with radar or sonar due to the highly-focused beam of a laser. For laser-based systems, optical cross-talk would only pose a serious issue in an environment full of highly polished surfaces. Indeed, more likely to occur is electronic cross-talk, which is the interference of signals within the circuitry.

Other surface features such as discontinuities can also cause reading uncertainties. If a laser beam is targeted directly on a surface edge, there is a possibility that part of the beam would be reflected back to the sensor while the remaining portion would continue until it reaches another surface before an echo is detected. This means multiple return signals would be received for one original transmission. This problem of conflicting range readings is known as the mixed pixel effect as explained by Hebert and Krotkov (1991). As such, it is important to keep the beam-width as narrow as possible. The inevitable divergence of a laser beam makes the problem more pronounced at long distances. Some filtering techniques have been proposed to deal with this problem, but their computational requirements render them too slow for real-time applications as explained by Tuley, Vandapel et al. (2005).

Environmental factors aside, the accuracy of laser range finders is ultimately dependent on the integrity of the electronic circuitry and the transmission stability of the laser diode.

In general, for depth-sensing purposes, the current view holds that the benefits of using laser outweigh its drawbacks. This explains the popularity of the technology. Currently, there are numerous types of laser range finders available. Most of them are based on one of the following three main principles: time-of-flight (TOF), triangulation and interferometry.

2.3.2.2 Other Types of Range Finders

Some other forms of depth-sensing systems employ similar principles to those behind the laser range finder. The main difference is in the transmitted signal. Instead of a coherent light found in laser, normal signals within the electromagnetic (EM) spectrum, such as infrared, microwave or radio waves, are emitted. The use of radio waves for range determination is commonly known as radar – an acronym for radio detection and ranging. Another popular depth detection technique uses mechanical waves in the form of sound. This latter method is also called sonar – short for sound navigation and ranging (Langer and Thorpe 1992). To prevent the sound signals from being an annoyance to human observers and bystanders, the emitted frequencies are usually in the ultrasonic range.

2.3.2.3 Laser Scanners

Range finders are found at the core of many laser scanners. Essentially, a scanner directs a range finder to sweep across a target area in order to gather a multitude of distance data points (Curless 1999). The type of illumination can be categorised into three broad groups. In one-dimensional illumination, a point beam is projected on the surface of an object. The beam is then swept in two dimensions across the target area. For two-dimensional illumination, the projected image is often a line. This requires scanning to be carried out in one dimension. Three-dimensional illumination employs structured light techniques, such as Moire patterns. No scanning is required as target area is illuminated all at once.

2.3.2.4 Range Determination by Time-of-Flight Method

In time-of-flight systems, the key measurement is the time taken for a transmitted signal to be reflected back from a target. Knowing the speed of light and compensating for the effects of temperature, pressure and humidity on the transmission medium, the distance of a target can easily be determined from the measured time-lag. TOF range finders tend to have good operating range, and it is not uncommon to find one that can measure over a kilometre. TOF systems can be further categorised into two general sub-types: pulse modulation and continuous wave (CW) modulation.

a. Pulse Modulation

This is the same time-tested principle behind the radar and sonar. When laser is employed as the transmitted signal, it is called lidar (Light Detection and Ranging) or ladar (Laser Detection

and Ranging). The idea behind pulse modulation is simple. A short signal pulse is transmitted while a detector awaits the echo rebounding from a target. While this technique is straightforward, some inherent difficulties exist.

The speed of light is approximately 300×10^6 m/s. For a TOF system, light has travel twice the distance between the sensor and object. So, for a distance of 1 m, the time taken for a pulse to be emitted and received is about 6.66×10^{-9} s. Such short timings require pulsed TOF systems to have highly precise electronic circuitry. Even an error of a nanosecond would lead to enormous inaccuracies. The problem becomes less acute as distance increases.

Another form of uncertainty arises from variations in the intensity of detected signals. The rise time of a return signal is dependent on signal strength, i.e. the greater the signal intensity, the quicker the peak is reached. Thus, for identical distances, systems that employ fixed threshold triggering would record a stronger signal earlier than a weaker one as demonstrated in Fig. 2.4. Since more reflective surfaces produce stronger return signals, they will appear to be nearer than less reflective ones. This discrepancy is called time-walk. A common remedy for this problem is the use of constant-fraction discriminators. Fig. 2.5 shows how detection is triggered on these devices when a predefined fraction of the peak amplitude is reached. However, as the value of the constant-fraction is set arbitrarily, this method can only be seen as an incomplete solution. The time-walk problem is compounded by the mixed pixel effect if there is a sharp transition between areas with a large difference in reflectivity.

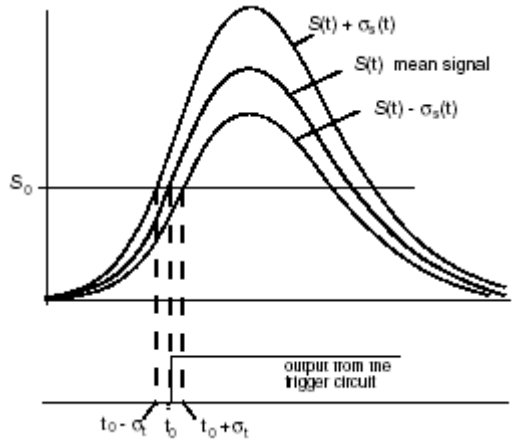


Fig. 2.4 Time-walk due to variation in signal intensity (Donati 2004)

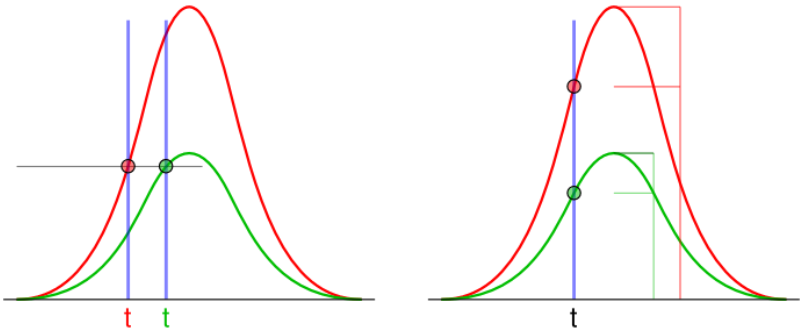


Fig. 2.5 Fixed threshold triggering versus constant fraction triggering (Dschwen 2006)

Ambiguity in signal detection can arise if a pulse is emitted before the return signal of a previous pulse is received. In such cases, it would be impossible to distinguish which return signal corresponds to which original pulse. If both return signals overlap, the resulting interference may render the readings useless. The time required for a signal to return depends on the distance. Hence, the pulse repetition interval must be adjusted to suit the range of the operational environment. If the time interval between consecutive pulses is too short, it would lead to ambiguous readings. On the other hand, a long pulse interval will result in a low rate of measurement.

b. Continuous Wave Modulation

In continuous wave (CW) modulation, the laser is transmitted via a carrier signal usually of lower frequency within the RF (radio frequency) band. When this modulated signal is reflected back from a target, it exhibits a shift in either its phase or frequency. This shift translates to a time-lag, which in turn is used to determine distance. Phase shift is measured using amplitude-modulated continuous wave (AMCW), while frequency shift is measured using frequency-modulated continuous wave (FMCW). CW systems are considered part of the TOF family because they ultimately measure the time interval between signal transmission and reception, albeit indirectly via phase or frequency shifts. As CW systems do not directly measure the travel time of a signal, they do not require the complicated high-speed electronics of pulsed systems.

i. Continuous Wave via Amplitude Modulation

AMCW systems transmit at the constant frequency of a carrier wave. The laser signal information is conveyed by variations in the signal strength of the carrier wave. As the

frequency of the carrier wave is already predetermined, a phase shift between the emitted and reflected signals can easily be converted to a corresponding time interval as shown in Fig. 2.6. However, a constant phase shift is repeated every cycle. Therefore, measurement of any distance beyond half the wavelength of the carrier frequency would lead to ambiguities. The ambiguity interval is half the carrier wavelength because the distance between the range finder and the target is half of the travel distance of the laser. In light of this important fact, it must be ensured that the range finder does not operate in an environment where there is a possibility of distances exceeding the ambiguity interval. To be certain, the carrier wavelength should be set in such a way where there would be plenty of allowance.

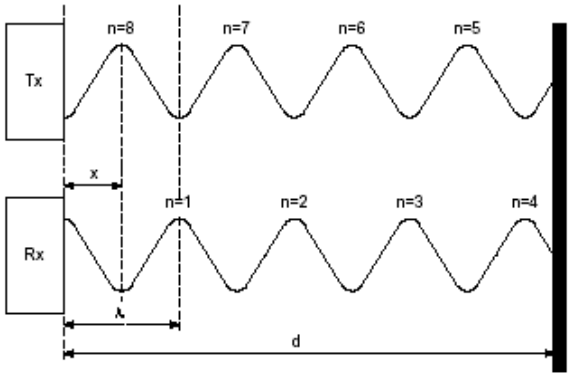


Fig. 2.6 Phase shift between transmitted and reflected signals (Everett 1995)

ii. Continuous Wave via Frequency Modulation

In FMCW systems, the laser signal information is transmitted by variations in the instantaneous frequency of a carrier wave while the carrier amplitude remains constant. Generally, the carrier wave is of the saw-toothed or triangular types. When a reflected signal is mixed with the original signal, the resulting frequency shift creates a beat frequency that is proportional to the range as illustrated in Fig. 2.7. Unlike AMCW systems, absolute distance can be measured with no ambiguity. However, FMCW systems are not without problems. Due to the fact that the laser signal is conveyed by variations in the carrier frequency, any fluctuation in the laser frequency itself can clearly affect the frequency of the modulated wave. As the response of a laser diode to a modulation frequency is seldom linear, and laser frequency is highly sensitive to temperature changes in the laser diode, signal stability is thus difficult to achieve. Even with the use of thermal control over the laser diode, signal drift cannot be completely eliminated. Difficulties in attaining linearity and design complexities explain why FMCW systems are quite uncommon.

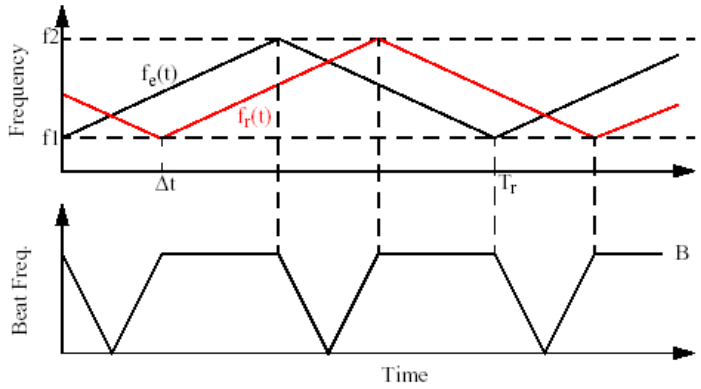


Fig. 2.7 Beat frequency of FMCW system (Hancock 1999)

2.3.2.5 Range Determination by Triangulation Method

The premise behind the triangulation technique is the utilisation of simple geometry to determine distance as opposed to measurement of elapsed time used by TOF systems. Using this principle, a wide variety of range-sensing techniques has been developed. A key drawback of the triangulation method is that measurement resolution decreases as range increases. Hence, this method is unsuitable for long range applications. Triangulation methods can be passive or active. Passive systems rely on ambient light and do not emit any radiation. Active systems are more versatile under different lighting conditions because such systems emit their own radiation.

An example of a passive method is stereoscopic photogrammetry. Images from two different angles of a single scene are captured and compared. Complex algorithms are then used to correlate image information and determine actual distance. Detection is difficult under dim lighting conditions and correlation is impossible if target surfaces have low-detail features or none at all. While such systems can achieve good accuracy, its complexity results in slow computation time and high cost. In general, this technique is more suited for limited-range applications that are stationary or slow-moving.

Other types of passive triangulation methods make use of focusing or defocusing effects. With a fixed focal length, the range is determined either by relying on active focusing or measuring the amount of blur on the focal plane. Since the focus/defocus techniques follow the laws of optics and have little resemblance to the conventional triangulation set-up, it may not seem obvious straightaway that the two principles are in any way related. However, closer inspection will show that these methods are geometrically similar as demonstrated by Pentland (1987), and Schechner and Kiryati (1998). The performance of range-from-focus systems rests on factors such as the optical uniformity of the lens and focusing capabilities.

Also, range resolution can be improved by using larger lenses, though that would add unnecessary weight. While the defocus method suffers from ambiguity problems, the focus technique does not. Unlike traditional triangulation arrangements where the emitter and detector are on separate axes, the lack of an emitter ensures that focus/defocus methods do not suffer from occlusion problems caused by the shading (also called shadowing) effect. This phenomenon will be explained a little later.

For active triangulation systems, the advantages of laser make it a popular choice in terms of applications. In a typical set-up, the baseline is the fixed distance between the laser emitter and detector. A virtual triangle is formed by connecting the baseline to the path from the transmitter to the targeted area, and from the targeted area to the receiver. The range from the baseline to the target is then easily worked out by using the law of sines as demonstrated in Fig. 2.8. Unlike pulse-modulated TOF systems, there is no necessity for the electronic circuitry to have capabilities to deal with very high sampling rates. Also, the stability of the emitted laser frequency is not an issue at all, in contrast to continuous wave techniques.

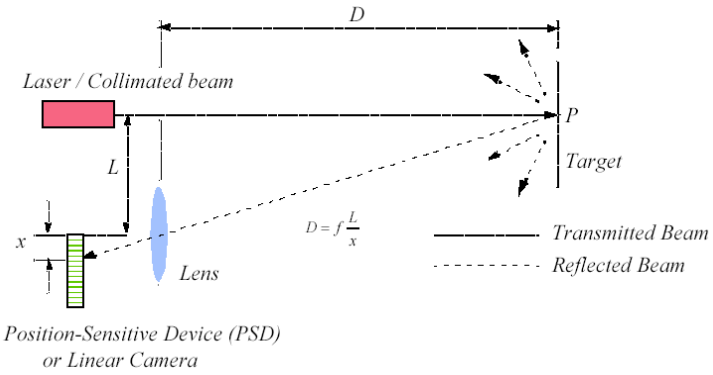


Fig. 2.8 Distance measurement by triangulation (Siegwart and Nourbakhsh 2004)

A common problem with the triangulation method is the shading or shadowing effect. The non-coaxial nature of the transmitter and receiver means that there will be occasions where features on a target surface can prevent the laser from illuminating certain areas. Similarly, features on a target surface can block the return path of a laser. These optical obstructions result in “shaded” areas that cannot be detected, such as shown in Fig. 2.9. The probability of shading occurrence can be reduced by narrowing the distance between the emitter and detector. However, this will come at the expense of measuring sensitivity and range. To alleviate this problem, Yoshida and Hirose (1988) have suggested using two detectors at the baseline with a laser transducer situated in between. Fig. 2.10 shows such a set-up. This provides some redundancy in case either of the detectors experiences occlusion. Other forms of multiple-detector systems have also been proposed (Pieper et al. 1995).

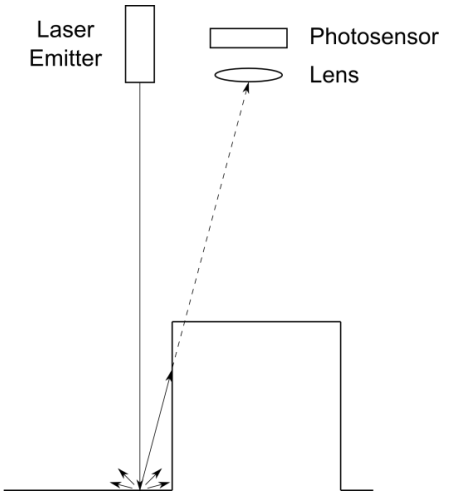


Fig. 2.9 Optical obstruction

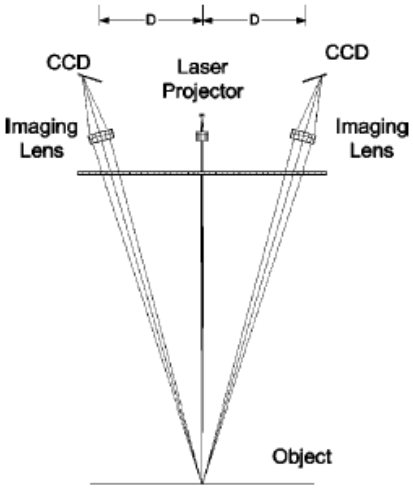


Fig. 2.10 Dual-sensor triangulation (Blais 2004)

The geometric dependency of the triangulation method means that the accurate detection of the location of a target spot is critical. When a laser beam lands on a highly uneven or extremely angled surface, the resulting spot image could be severely distorted. An example of such a situation is shown in Fig. 2.11. The difficulty in determining the centre of the beam would lead to measurement uncertainties not dissimilar to the mixed pixel problem. Distortion in the spot image can also arise from random intensity fluctuations. This phenomenon is called speckle, and is caused by interference within a highly coherent signal such as laser. Fig. 2.12 shows an example of speckle. Numerous filtering algorithms have been proposed to mitigate speckle noise.

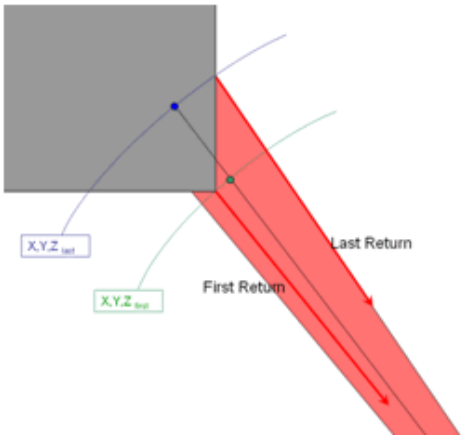


Fig. 2.11 Uneven beam focus spot (Center for Advanced Spatial Technologies.)

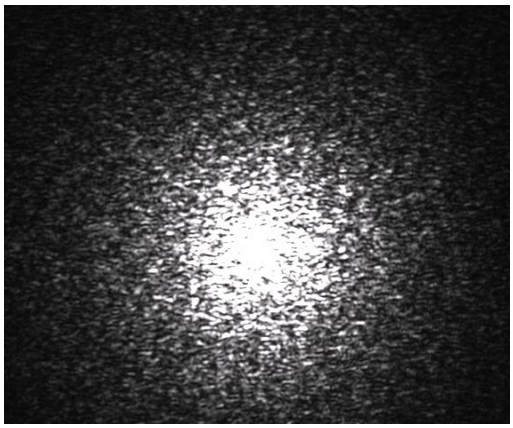


Fig. 2.12 Speckle (Bilenca)

The limited range of triangulation systems is not solely due to the fact that there is a loss in measurement sensitivity with increasing distance. The non-coaxial arrangement of the transmitter and detector means that a laser spot beyond a certain range may not be visible within the detector's field of view. For most current sensors, the field of view is sufficiently wide for medium-range spot detection. However, it is a restricting factor in laser scanning operations that utilises 2D or 3D illumination.

For short ranges, range finders operating on the triangular technique generally tend to provide higher measuring accuracy than TOF systems.

2.3.2.6 Range Determination by Interferometry Method

Conventional laser interferometry shares the same principle with modulated continuous wave techniques. The key difference is the lack of a carrier signal in conventional interferometry. In this technique, a single laser beam is split into a reference beam and a measuring beam. The reference beam is sent immediately to a fringe detector while the measuring beam is projected onto a target and is then reflected back to the detector. The two beams are then optically recombined. The resulting is a series of interference fringes. A change in object distance would lead to a new interference pattern. The relative distance between the current and previous positions of an object can be determined by counting the interference fringes (Yatsenko et al. 2004). The extremely short wavelength of light provides high measuring resolution. However, this method suffers from ambiguity problems similar to the AMCW method. Due to the exceedingly minute wavelength of light, the absolute measuring range is impractically small. Nevertheless, this technique could be used to determine relative distance with tremendous accuracy, provided that the device components possess high stability.

2.3.3 Signal Beacon or Transponder Systems

Other than the sensor systems mentioned above, there are numerous other kinds in use. One relatively common type is the signal beacon system. Transmitted signals are often radio waves, infrared or laser. For ground-based radio-frequency (RF) systems, there are two main types: active and passive.

Active beacon systems work very differently from their passive counterparts. For the initial calibration, the subject and various transponders are situated at known locations. The vehicle then sends out interrogating signals to three or more reference transponders which in turn reply respectively. This is to determine the time needed for each transponder to reply after receiving a signal from the vehicle. This response lag is also called the turn-around delay

(TAD). After the calibration process is complete, the calculation of the absolute distance between the vehicle and a particular transponder is a simple matter.

The total elapsed time between an initial signal broadcast and the reception of a response from a transponder can be readily documented. Subtract the TAD from the total elapsed time, and the result is the time required for a signal to travel from the vehicle to a transponder and back. Along with the knowledge that the propagation speed of radio waves is no different from that of light, the distance between the vehicle and transponder can easily be calculated. This will give a circular locus of possible positions of the subject. With three or more transponders, the intersection of the corresponding loci will provide the location of the vehicle. This method is called trilateration, and it measures the absolute time of arrival (TOA) of various signals in order to determine a subject's location. However, it will be shown later that trilateration can apply to passive systems as well.

Passive beacon systems await signals broadcast simultaneously from known transmitter sites with accurate locations. The various distances between the subject and the transmitters will result in the signals arriving at different times. Since the time taken from broadcast to reception is not known, the absolute distances between the vehicle and the transmitter sites cannot be directly established. Only the time difference of arrival (TDOA) can be determined. This is done by noting the delay in the reception of the various signals relative to one another or analysing the phase differences among them. The result is a hyperboloid locus of possible positions. Using four transmitter stations, three sets of TDOA data will be available, and the resulting intersection of the loci will provide the position of the vehicle. This method is also known as multilateration or hyperbolic positioning.

A large-scale example of a trilateration positioning system is the Global Positioning System (GPS) developed by the US Department of Defence. This system has 31 (formerly 24) satellites that orbit the earth roughly every 12 hours at an elevation of about 10,900 nautical miles. The positions of the satellites are tracked by ground-based stations at all times. This system differs from typical ground-based counterparts in that the satellites and receivers all have synchronised internal clocks. Signals sent by each satellite contain the time of transmission. When a signal is received, the receiver notes the time of reception and calculates the total elapsed time for the signal to travel between the satellite and itself. The distance can then be determined by using the trilateration technique. Since the receivers only receive signals and do not transmit, the system is passive.

Many factors can affect the accuracy of GPS. Atmospheric effects can lead to refraction or dispersion of signals. As a result, signal speeds may be retarded and lead to erroneous calculation of distances. Multi-path effects are another potential source of error. A receiver

can be confused by direct and reflected signals. Synchronisation errors among the satellites and receivers are also a possible cause of inaccurate readings.

For civilian purposes, the nominal precision of GPS is limited to about 15 m. According to the most recent report by the US Department of Defense regarding GPS, its global average accuracy ranges from 6.0 m to 12.8 m based on 95% of reported data (USAF 2008). This accuracy can be enhanced to a range of less than 1 m by relying on additional ground-based reference stations or supplementary GPS receiver units positioned at precisely-known locations. However, in order to take advantage of the augmented GPS precision, expensive equipment is required. Moreover, the accuracy of systems that use additional ground-based beacon stations deteriorate as distance increases between the base stations and receiver unit (Monteiro et al. 2005). For large operational area, this level of precision may be sufficient. However, for most smaller-scale applications, the available degree of accuracy is often inadequate. Furthermore, unless there are signal repeaters installed within a building, GPS cannot function indoors or in other sheltered or enclosed places such as tunnels.

Another well-known positioning method is triangulation. Unlike trilateration, where the location of a position is derived from direct measurement of distances, triangulation depends on the determination of angles made between points to known baselines. Examples include beacons situated at known locations which emit signals such as infrared light. When within range, the scanning sensors on a robot will each detect a different angle of incidence relative to a pre-determined baseline. Using simple geometry, the distance of the vehicle with respect to each beacon can be determined.

The biggest drawback of signal beacon systems is the need for setting up beacons as well as measuring their locations precisely before any tracking activity can commence. Also, the effective range of the beacons and the corresponding placement issues often limit the flexibility such systems.

2.4 Control Systems

The proportional-integral-derivative (PID) controller is the oldest and most common type of control-loop feedback system used for industrial control (Bennett 2001). For decades, PID controllers, along with PI and PD variants, are the de facto mechanisms used in process control. Generally easy to tune, especially with the aid of established methods, it provides a relatively quick and convenient way to automate a control process. Hence, the PID controller remains a popular choice to this day. For the same reason, PID controllers are also regularly found on robotic systems.

Numerous tuning techniques for PID controllers have been introduced over the years, such as those based on heuristics, root-locus, pole-placement, frequency response, step response, integral error optimisation, etc. (Åström and Hägglund 1995; Tan et al. 1999; Tan et al. 2006). Out of all the methods, the heuristic approach of Ziegler-Nichols is one of the most popular. Another well-known empirical approach is the Cohen-Coon method, although it is not as widely-used as Ziegler-Nichols. For cascaded PID systems, there are no standard or widely-recognised tuning techniques (Lee et al. 1998; Sadasivarao and Chidambaram 2006; Leva and Marinelli 2009). So, the tuning procedure is generally quite a bit more complicated than non-cascaded systems. Naturally, the complexity is increased when multiple variables are involved.

The main purposes of the proportional action are to reduce rise time and steady-state error. If the value of this parameter is set too high, overshoot would result and the system response may oscillate. On the other hand, if this parameter is set too low, the controller may not be sufficiently responsive to large errors.

The integral action reflects the accumulated error over time and is mostly used for eliminating steady-state error. Thus, it has a correlation with past error values. It also helps in reducing rise time without the need for larger proportional gain. If the gain setting for this parameter is excessive, it would lead to overshoot and increased oscillation.

The derivative action exhibits a damping effect and is used for slowing the rate of change of the response. This helps reduce overshoot and settling time. By varying the rate of change of the signal, this parameter has a predictive effect on the resulting output. This is in contrast to the reactive nature of the integral action. This parameter is highly sensitive to noise. If set too aggressively, the damping effect would be lost and instability would ensue.

The effectiveness of PID controllers can be extended by having a set of tuned settings for each different operating range - as determined by the observable variables. This approach is called gain scheduling, and ensures that control is almost always optimal. In general, conventional PID controllers are unsuitable for systems that are highly non-linear and unstable.

In addition to PID controllers, there are many other types of mechanisms that provide some form of automated control, such as self-tuning regulators (Åström and Wittenmark 1973; Åström et al. 1977), robust (Zhou and Doyle 1998) and adaptive (Sastry and Bodson 1994; Dumont and Huzmezan 2002; Pourboghrat and Karlsson 2002) controllers. While the general ideas of robust and adaptive control are similar, there is a key difference between the two. In robust control, the limits of uncertainties in the parameter or frequency domains must be

specified a priori. In contrast, there are no such restriction in adaptive control (Landau et al. 2011).

All the general approaches in control theory, such as ones mentioned above, utilise various methods to achieve their goals. These include sliding mode (variable structure) control, linear quadratic regulator, Lyapunov's method, etc. The latter theory (sometimes spelt Liapunov) was proposed near the end of the 19th century, and was largely overlooked for the next 50 years (Parks 1992). The stability principle that arose from the theory has now become an indispensible part of the knowledge base in field of control systems.

There are two methods proposed by Lyapunov. The first method examines the local stability of a system that has been linearised and requires the solution of differential equations of the system (Murray et al. 1994). It is also called the indirect method. The second method checks the asymptotic or global stability of an equilibrium point without the need for solving differential equations of the system (Slotine and Li 1991; Khalil 2002). An additional advantage over the first method is that it can be used on systems that cannot be linearised. This approach is also called the direct method and is used much more widely than the first method.

This thesis shall adopt the technique of model reference adaptive control (MRAC) based on the direct Lyapunov stability method (Gholipour and Yazdanpanah 2003). MRAC is also known as MRAS, which stands for model reference adaptive system. This is a concept where the output of a system is constantly compared to the expected response of reference model. The updatable parameters of its closed-loop controller are then adjusted accordingly to match the reference model's response (Slotine and Li 1991).

2.5 Summary

After considering the various types of sensor technologies and methodologies, it was decided that a tracking system based on the optical mouse sensor was the best option for the purpose of this research. This type of sensor has extremely high precision and requires very little processing power and virtually no maintenance. Furthermore, it is cheap and widely available. A WMR that uses the optical mouse sensor as a tracking device would not require the use of any external equipment (e.g. transponders), so deployment would be relatively simple. It will also be able to function indoors. As for the choice of the control system, both PID and adaptive systems would be explored.

Chapter 3: General System Analysis and Design

The overarching idea behind most autonomous mobile robots is simple. First and foremost, the robot must be able to locate its own position within a pre-defined frame of reference. This positional determination process is called localisation. Next, a desired path or destination is defined within the same reference frame. Once the robot is able to ascertain its own current location, attempts will be made to propel it towards its target course or position. As the robot moves towards its goal, its location will be constantly compared with that of its objective, and adjustments in speed and heading will be made as and when necessary. This process will be repeated until the final objective is achieved.

In this chapter, the general mathematical model of the differentially-driven wheeled robot will be developed and analysed. This will include the exploration of both kinematic and dynamic systems along with motor and friction modelling.

3.1 Mathematical Modelling

For this study, a frame of reference must first be defined before the WMR can carry out its task. This reference frame is based on the Cartesian coordinate system and should take into account the entire area of operation. Once this is complete, the initial positional data from the tracking sensors can be initialised with respect to the reference frame.

One of the main objectives of this study is the design of a WMR that is relatively uncomplicated. This effectively rules out the use of point-to-point stabilisation as a control strategy. The trajectory-tracking approach was selected in favour of the path-following scheme because it allowed for more control over the speed of the WMR.

The ability of the WMR to determine its own location within a reference frame largely depends on a pair of optical sensors. These optical sensors are similar to the ones typically found in computer mice. Here, the sensors essentially act as the “eyes” of the WMR by “looking” at the surface of travel and recording the distance moved from one time sample to the next. Both sensors pick up positional data in the same manner. However, one is used for localisation while the other serves to establish the WMR’s heading - the direction in which it is pointed. This information is determined by analysing the relationship between the positional data provided by the two sensors.

In this thesis, the mathematical modelling of the WMR will be addressed in two broad sections: kinematics and dynamics. The kinematic equations define the motion of the WMR while the dynamic equations outline its operational limits. The system is fundamentally non-linear.

The approximate mathematical model of the differentially-steered WMR used here is a rectangular box with two independently-driven rear wheels and a single front castor wheel as shown in Fig. 3.1. For simulation purposes, the nominal overall dimensions of the WMR model sans wheels are 0.2 m, 0.15 m, 0.1 m for length, width and height respectively. The radius of the wheels is 0.03 m. The total weight of the WMR is 1.5 kg.

For positioning and navigation, the x-y global coordinate system has its origin labelled O, while the x'-y' local coordinate system has its origin located at the centre of gravity, G, of the WMR.

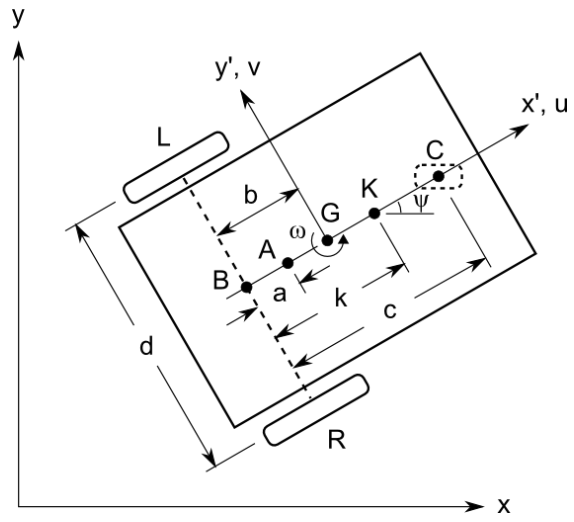


Fig. 3.1 Differentially-steered WMR

The key points of the model are the centre of the wheel baseline, centre of gravity, locations of dual sensors, and points of contact of the left, right and front wheels with the ground. These are represented by B, G, A, K, L, R and C respectively.

3.1.1 General Assumptions and Limitations

Wheel (or tyre) slippage often contributes significantly to the discrepancy between the expected and actual positions of a moving vehicle. This form of tracking error is considered to be non-systematic and difficult to model. As mentioned in earlier chapters, the optical tracking sensors used here have the ability to track the location of the WMR accurately - regardless of slippage. The control algorithm simply factors any positional error into the calculations, and

necessary adjustments are then made accordingly. In other words, no-slip conditions can be assumed in the calculations while actual slippage is treated as part of the feedback error. Hence, while frictional limits of the wheels are an important consideration for keeping potential slippage to a minimum, the modelling of wheel slippage is not crucial for this study.

The tracking sensors on the WMR are accurate and efficient if used within operational limits. In regards to the images captured by the sensors, it is important that the rotation from one frame to the next be kept below the maximum allowable rate. Otherwise, it could lead to erroneous pixel-matching of sensor images between frames (Singh and Waldron 2004).

There are other factors that can also prevent the proper operation of the tracking sensors. Undulating terrains affect the angles of incidence and reflection of the laser beam, and can result in inaccurate location readings. Hence, these sensors work best on flat travelling surfaces. Additionally, completely featureless or mirror-like surfaces can prevent these sensors from tracking properly. Also, puddles of liquid on the ground can form ripples that distort the tracking laser. A dusty or smoky atmosphere may diffuse or even obstruct the tracking laser as well.

3.2 Kinematic Modelling

The behaviour of a WMR is fundamentally governed by kinematics. The distance between checkpoints and the time allocated for travel determine the velocity of the WMR and, by extension, its wheel speed. In contrast, dynamics play a less direct albeit still important role in this study. Its function is to ensure that the WMR does not exceed operational limits.

3.2.1 Assumptions and Limitations

Dynamic factors are not taken into account in this section. Thus, inertial effects such as weight transfer, as well as friction do not play a part in the calculations.

3.2.2 Kinematic Equations

Referring back to Fig. 3.1, the wheel radius, track width, and distance between the wheel baseline and centre of mass are respectively identified as r, d and b. The left and right wheel angular velocities are ω_l and ω_r, respectively. The longitudinal, lateral and yaw (turning) velocities are respectively labelled u, v and ω, and the yaw angle is ψ.

Taking the centre of mass as the representative point of the WMR, the equations of motion in terms of the angular velocity of the driving wheels can be described as follows:

$$u = \dot{x}' = r\,\frac{(\omega_l + \omega_r)}{2} \tag{3.1}$$

$$v = \dot{y}' = \frac{br}{d}\,(\omega_r - \omega_l) \tag{3.2}$$

$$\omega = \frac{r}{d}\,(\omega_r - \omega_l) = \frac{v}{b} \tag{3.3}$$

Eqs (3.1) and (3.2) can be rearranged to obtain the following:

$$\omega_l + \omega_r = \frac{2u}{r}$$

$$\omega_r - \omega_l = \frac{dv}{br}$$

$$\therefore \quad \omega_l = \frac{1}{r}\left(u - \frac{dv}{2b}\right) \tag{3.4}$$

And $$\omega_r = \frac{1}{r}\left(u + \frac{dv}{2b}\right) \tag{3.5}$$

The reference point of the WMR's position is at sensor A, and not at mass centre G. In the global coordinate system, the equations of motion with respect to the tracking sensor are:

$$\dot{x} = u\cos\psi - \left[v + (a-b)\,\omega\right]\sin\psi \tag{3.6}$$

$$\dot{y} = u\sin\psi + \left[v + (a-b)\,\omega\right]\cos\psi \tag{3.7}$$

$$\dot{\psi} = \omega \tag{3.8}$$

But $v = b\omega$,

$$\dot{x} = u\cos\psi - a\omega\sin\psi \tag{3.9}$$

$$\dot{y} = u\sin\psi + a\omega\cos\psi \tag{3.10}$$

By resolving Eqs (3.6) and (3.7), the local velocities of the WMR in the global coordinate system can be written as follows:

$$u = \dot{x}\cos\psi + \dot{y}\sin\psi \tag{3.11}$$

$$v = -\dot{x}\sin\psi + \dot{y}\cos\psi - (a-b)\,\omega \tag{3.12}$$

$$\omega = \frac{-\dot{x}\sin\psi + \dot{y}\cos\psi}{a} \tag{3.13}$$

$$v = \frac{b}{a}\left(-\dot{x}\sin\psi + \dot{y}\cos\psi\right) \tag{3.14}$$

The positional data of the second sensor K is processed in a similar way to sensor A by swapping the variable a for k.

For a WMR that uses one optical mouse sensor, the calculations of positional and velocity information are based on kinematic equations that have non-holonomic constraints. While the sensor may be able to pick up lateral movement caused by slippage, the data could be incorrectly processed if the kinematic equations are used. This is because the equations are based on the possible range of motion of a non-holonomic vehicle without any slippage.

The addition of a second sensor allows the calculations to be based on actual geometry and be free of non-holonomic limits present in the kinematic equations. This means that a full range of motion can be considered and accurately accounted for.

Since both sensors are situated at a fixed distance from each other along the longitudinal axis of the vehicle, they will cover the same distance along that axis. Hence,

$$\Delta x_1' = \Delta x_2' \tag{3.15}$$

Using simple geometry as shown in Fig. 3.2, the yaw angle based on data from two sensors can be determined by:

$$\psi = \sin^{-1}\left(\frac{\Delta y_2' - \Delta y_1'}{L}\right) \tag{3.16}$$

$$L = k - a \tag{3.17}$$

It must be noted that the optical mouse sensor does not distinguish between a straight line and a circular arc of the same length. Since a change in vehicle heading requires angular (yaw) movement that follows a curved path, the straight-line distances represented by Δy_1 and Δy_2 in Fig. 3.2 are actually respective arcs of identical lengths. This is explained in more detail in Chapter 6. Therefore, Eq (3.16) should be rewritten in a more accurate polar form as:

$$\psi = \frac{\Delta y_2' - \Delta y_1'}{L} \tag{3.18}$$

The turning velocity of Eq (3.13) can now be revised as such:

$$\omega = \dot{\psi} = \frac{d}{dt}\left(\frac{\Delta y_2' - \Delta y_1'}{L}\right) \tag{3.19}$$

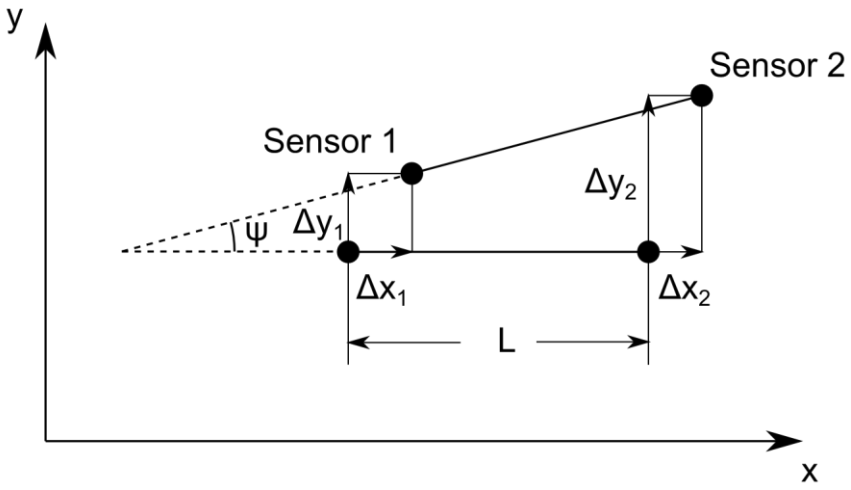


Fig. 3.2 Simple geometric representation of sensor positions

It is obvious that the simplified model shown in Fig. 3.2 is an approximation and not a totally accurate depiction of the actual geometric motion of the WMR. Due to the fact that the sensors' positions can only be measured at prescribed intervals, discretisation errors are thus unavoidable. However, these errors are usually manageable and often insignificant when the time period between samples taken is very small, i.e. sampling frequency is high.

Figure 3.3 illustrates a pair of sensors in two poses beyond the initial position as well as their respective approximated positions. These poses represent the sensors' positions after one sample period with two different sampling rates used. The first shows the sensors being sampled at a shorter time interval and is associated with angle ψ_1. The second shows the sensors being sampled at a later time compared to the first due to the use of a longer sampling period, and is associated with angle ψ_2. This is an example of a first-order Euler approximation.

In the diagram, the discrepancy between actual and approximate positions associated with the shorter sampling period is not marked out because the error is quite small. In the case of the longer sampling period, the discretisation error is quite significant and can be clearly seen. It is evident from the illustration that discretisation error increases as the sampling period lengthens, i.e. sample frequency decreases. Conversely, discretisation error approaches zero as the sampling period shrinks. Being that dead-reckoning errors are cumulative, it is

imperative that the sampling frequency used should be as high as computationally feasible in order to preserve an acceptable level of accuracy.

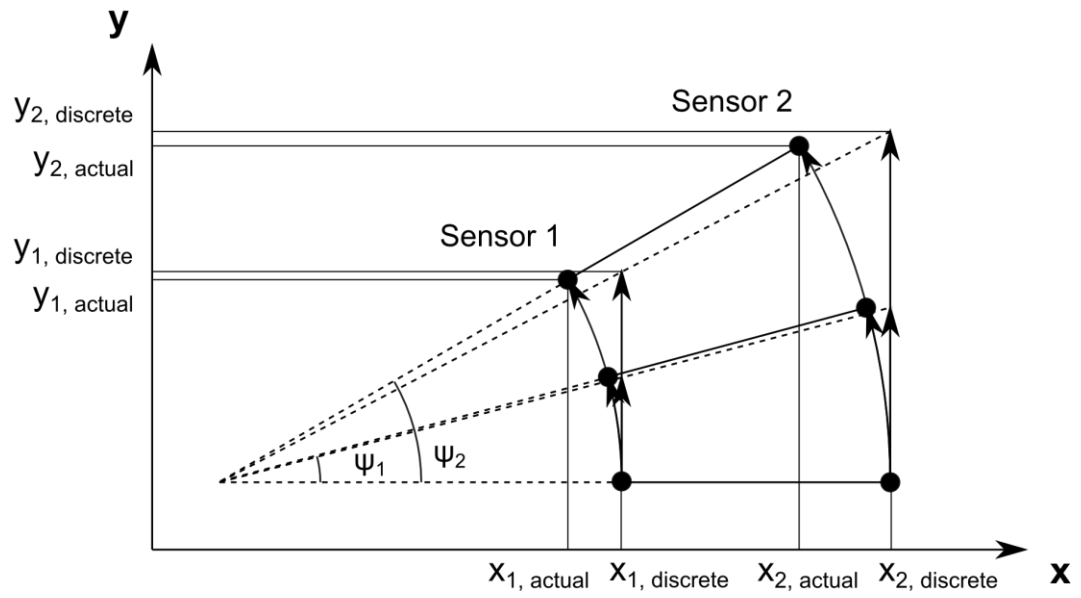


Fig. 3.3 Approximation error due to discretisation

3.3 DC Motor Modelling

The propulsion of the WMR is largely dependent on a pair of DC motors – one for each wheel. Both the velocity and direction of the WMR are controlled by simply varying the speed of each motor. To work out the necessary voltage to drive the motors at the required rotational speeds, inverse kinematic calculations are used to simulate the motor controller.

Referring to Kuo (1995), let T_m, K_t and i_a represent the motor torque, torque constant and armature current respectively. Since the motor torque is proportional to the current flowing through the armature, their relationship can be established as:

$$T_m(t) = K_t \; i_a(t) \tag{3.20}$$

The angular displacement of the rotor θ_m, back-emf E_b, and back-emf constant K_b are related to each other in the following manner:

$$E_b(t) = K_b \; \frac{d\theta_m(t)}{dt} \tag{3.21}$$

Additionally, the armature resistance, armature inductance, applied voltage, rotor inertia, viscous-friction coefficient and load torque are denoted as R_a, L_a, E_a, J_m, B_m and T_L respectively. They can be written in terms of Kirchhoff's and Newton's laws as such:

$$L_a \frac{di_a(t)}{dt} + R_a\ i_a(t) = E_a(t) - E_b(t) \tag{3.22}$$

$$J_m \frac{d^2\theta_m(t)}{dt^2} + B_m \frac{d\theta_m(t)}{dt} = T_m(t) - T_L(t) \tag{3.23}$$

The transfer function between the applied voltage and rotor speed is acquired via the use of Laplace Transforms:

$$\frac{\dot{\theta}_m(s)}{E_a(s)} = \frac{K_t}{(L_a s + R_a)(J_m s + B_m) + K_t\ K_b} \tag{3.24}$$

The above equation can be represented as a block diagram as shown in Fig. 3.4. It is important to note that in SI units, the values of K_t and K_b are exactly equal.

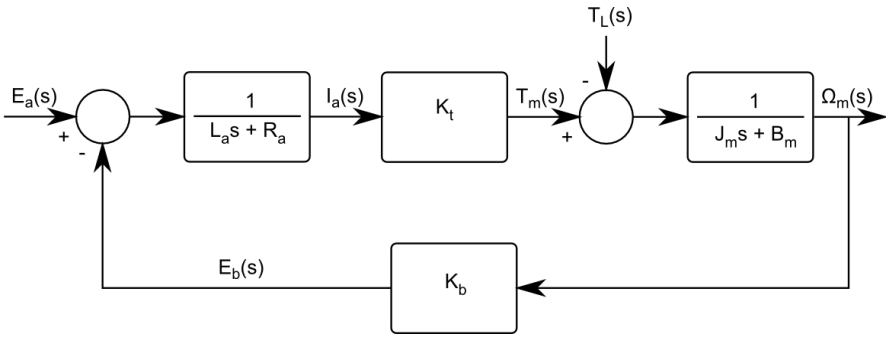


Fig. 3.4 Block diagram of a DC motor system

3.4 Dynamic Modelling

Modelling a WMR's behaviour on kinematics alone will likely lead to inaccuracies especially at increasing mass and velocity. To prevent slippage, dynamic forces have to be taken into account. For the WMR modelling employed in this study, dynamic constraints are imposed on kinematic solutions in order to produce realistic results.

3.4.1 Assumptions and Limitations

The point of reference for dynamic considerations is usually the centre of mass. However, uneven mass distribution can often contribute to significant dynamic effects for which are unaccounted. Hence, neglecting substantial mass concentration can severely affect the accuracy of a mathematical model. In this study, the WMR is not deemed to have any areas of significant mass concentration.

Friction is an important dynamic factor. Rolling friction without slippage is essentially static friction. When slippage occurs, dynamic friction replaces static friction. Since the coefficient of dynamic friction is usually lower than that of static friction, the former coefficient is used here as a conservative estimate for calculating acceleration limits before the onset of slippage. All forms of friction, whether static, dynamic or viscous, are assumed to be constant in this report. Also, the frictional characteristics of the wheels (tyres) are taken to be uniform in all directions.

It has been mentioned in an earlier section that the travelling surface has to be relatively flat in order for the optical tracking sensors to work accurately. Also, a non-level surface can increase the tendency of the WMR to topple. In such a situation, no acceleration adjustments can be made to prevent it from tipping over as there are no sensors on the WMR to measure its vertical posture.

3.4.2 Dynamic Equations

The motion of the WMR model is restricted to three degrees of freedom. Taking the WMR as reference, forces occur in the plane consisting of both lateral and longitudinal axes. A turning moment causes rotation in the perpendicular axis. The mass and moment of inertia are labelled m and I_z respectively, while forces exerted by the driving wheels are denoted by $F_{x'l}$, $F_{x'r}$, $F_{y'l}$ and $F_{y'r}$.

$$F_{x'l} + F_{x'r} = m\left(\dot{u} - v\omega\right)$$

$$\therefore \quad \dot{u} = \frac{1}{m}\left(F_{x'l} + F_{x'r}\right) + v\omega \tag{3.25}$$

$$F_{y'l} + F_{y'r} = m\left(\dot{v} + u\omega\right)$$

$$\therefore \quad \dot{v} = \frac{1}{m}\left(F_{y'l} + F_{y'r}\right) - u\omega \tag{3.26}$$

$$\dot{\omega} = \frac{1}{I_z}\left[\left(F_{x'l} - F_{x'r}\right)\frac{d}{2} - \left(F_{y'l} + F_{y'r}\right)b\right] \tag{3.27}$$

Revisiting Eqs (3.1) to (3.3), the above acceleration Eqs (3.25) to (3.27) can also be written in terms of the rotational acceleration of the driving wheels. The yaw (turning) acceleration of the WMR is known here as α.

$$\dot{u} = r\,\frac{\left(\dot{\omega}_l + \dot{\omega}_r\right)}{2} \tag{3.28}$$

$$\dot{v} = \frac{br}{d}\left(\dot{\omega}_r - \dot{\omega}_l\right) \tag{3.29}$$

$$\dot{\omega} = \frac{r}{d}\left(\dot{\omega}_r - \dot{\omega}_l\right) = \frac{\dot{v}}{b} \tag{3.30}$$

$$\alpha = \dot{\omega} \tag{3.31}$$

3.4.3 Friction Modelling

To prevent slippage, the maximum allowable force on each wheel is its frictional limit. This means that for any given moment, the resultant of forces on each wheel in both lateral and longitudinal axes cannot exceed the maximum allowable force without slippage.

$$F_{\max} = ma_{\max} = \sqrt{F_x{}^2 + F_y{}^2} \tag{3.32}$$

The above equation is the basis of a simple and popular friction model. It also happens to describe a circular locus and is therefore commonly referred as the friction circle. The equation can be rewritten in terms of acceleration by factoring out the mass.

$$a_{\max} = \sqrt{a_x{}^2 + a_y{}^2} \tag{3.33}$$

Denoting N as the force that is normal to the frictional plane, and μ as the friction coefficient, frictional force is generally defined as:

$$F_{\max} = \mu N \tag{3.34}$$

On a level surface, the normal force is none other than the weight of an object itself.

$$N = mg \tag{3.35}$$

The maximum force that can be applied without slippage would be:

$$F_{\max} = \mu mg \tag{3.36}$$

Referring to Beckman (1991), let p(i) be the fraction of the WMR's mass supported by a particular wheel at any given time. The frictional limit on the wheel would now be:

$$p(i)ma_{\max} = \mu\, p(i)mg \tag{3.37}$$

This equation reduces to:

$$a_{\max} = \sqrt{a_x^{\,2} + a_y^{\,2}} = \mu g \tag{3.38}$$

The above clearly demonstrates that mass is not a factor in the determination of the maximum allowable acceleration without slippage. This means that weight transfer has no bearing on acceleration limits. Only the coefficient of friction matters in this case.

3.4.4 Dynamic Constraints

The friction model explored in the previous section is crucial to the determination of the dynamic constraints. Essentially, the velocity and acceleration of the wheels of the WMR are constantly checked to ensure that the resultant forces on the WMR will not cause it to exceed frictional limits.

A typical curved path segment of the WMR is shown in Fig. 3.5. Travelling along the path, the WMR makes an angular displacement θ. At the same time, the left and right wheels cover distances s_l and s_r respectively. The path radii for the left and right wheels are respectively represented by $r_{path,\,l}$ and $r_{path,\,r}$, and the track width of the WMR is denoted by d.

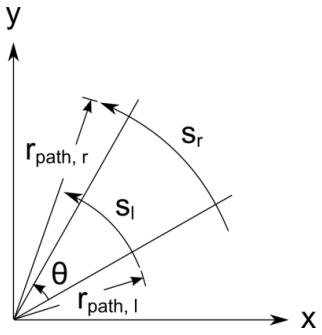


Fig. 3.5 Curved path segment of WMR

The relationship between the two path radii is defined by:

$$\left| r_{path,\,r} - r_{path,\,l} \right| = d \tag{3.39}$$

Displacements for both wheels are:

$$s_l = r_{path,\,l}\ \theta \tag{3.40}$$

$$s_r = r_{path,\,r}\ \theta \tag{3.41}$$

The left and right wheels, each of radius r, rotate at angular velocities ω_l and ω_r. For a given time t, the wheel displacements can also be written as:

$$s_l = r\omega_l\ t \tag{3.42}$$

$$s_r = r\omega_r\ t \tag{3.43}$$

By resolving Eqs (3.36) to (3.39), the path radius of each wheel can be represented in terms of the wheel angular velocity:

$$\frac{\omega_l}{\omega_r} = \frac{r_{path,\,l}}{r_{path,\,r}}$$

$$\frac{\omega_l}{\omega_r} = \frac{r_{path,\,l}}{r_{path,\,l} \pm d}$$

$$\therefore \qquad r_{path,\,l} = \pm \frac{\omega_l}{\omega_r - \omega_l}\, d \tag{3.44}$$

And $$r_{path,\,r} = \pm \frac{\omega_r}{\omega_r - \omega_l}\, d \tag{3.45}$$

For a WMR turning at angular velocity ω, the normal acceleration a_{nb} at the centre of the wheel baseline is equal to the centripetal acceleration.

$$a_{nb} = \frac{r_{path,l} + r_{path,r}}{2} \omega^2 \tag{3.46}$$

This normal acceleration is also the same as that at the left and right wheels, respectively denoted as a_{nl} and a_{nr}.

$$a_{nl} = a_{nr} = a_{nb} \tag{3.47}$$

For a WMR turning at angular acceleration α, the tangential acceleration a_{tl} of the left wheel would be:

$$a_{tl} = r_{path,l} \alpha \tag{3.48}$$

For the left wheel with radius r and angular acceleration α_l, the above can also be written as:

$$a_{tl} = r\alpha_l \tag{3.49}$$

Similarly, for the right wheel with angular acceleration α_r, the tangential acceleration a_{tr} would be:

$$a_{tr} = r_{path,r} \alpha \tag{3.50}$$

Or $$a_{tr} = r\alpha_r \tag{3.51}$$

The resultant accelerations a_l and a_r respectively at the left and right wheels are:

$$a_l = \sqrt{{a_{nl}}^2 + {a_{tl}}^2} \tag{3.52}$$

$$a_r = \sqrt{{a_{nr}}^2 + {a_{tr}}^2} \tag{3.53}$$

3.5 Summary

A detailed mathematical model of the wheeled robot has been developed and presented in this chapter. Both kinematic and dynamic systems were examined, and the respective assumptions and limitations were duly noted. For the latter system, a simple friction model was also introduced. The Euler approach was the chosen method for localisation here. Its shortcomings are well known and this was clearly illustrated along with an explanation on how the issue can be managed. This theoretical model will serve as a basis for the computational model to be implemented in the following chapter.

Chapter 4: Computational Design of the PID-Based Model

Before a prototype is built, computer simulation is first performed to verify the validity of the mathematical model as well as the control algorithm. This step helps to reduce the amount of construction and alterations of prototypes needed, and thus saves time and resources. Based on the theoretical model developed previously, a computational representation will be formulated in this chapter to include both kinematic and dynamic algorithms as well as a PID control system. The entire simulation will be conducted by using the MATLAB® program and its graphical modelling tool, Simulink®.

4.1 General Outline of the Computational Model

The idea behind the overall design is quite basic as can be seen in Fig. 4.1. A set of checkpoints forms a desired path for the WMR. It then has to arrive at each checkpoint at a preset time. The two sensors provide the heading and coordinates of the current location. With the knowledge of the current position and next checkpoint, as well as the allotted time for the WMR to travel between the two locations, the direction and velocity can be determined. Using inverse kinematics, the instantaneous angular velocity of the driving wheels can now be calculated.

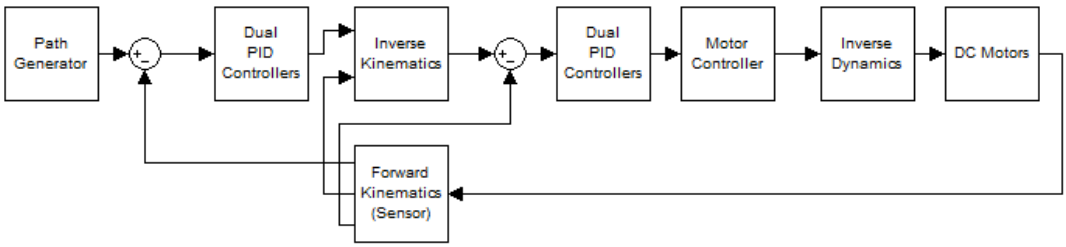


Fig. 4.1 PID-Based Tracking Control System of the WMR

The rotational speeds of the wheels are directly dependent on how fast the motors are spinning. The operation of the motors is in turn governed by a motor controller. Knowing the required angular velocity of the wheels, the motor controller can compute the necessary voltage for each motor.

Before the motor controller sends a voltage signal to each motor, a check has to be made to ensure that the motors do not turn overly fast to the point where the wheels lose traction longitudinally or laterally. With the application of inverse dynamics, the maximum allowable

velocity for each wheel can be determined from the frictional limits of the wheels. If limits are exceeded, motor speeds on both sides will have to be scaled back proportionally, and the motor controller will need to reduce voltages accordingly. In a purely kinematic model under no-slip conditions, the procedure for the verification of dynamic limits is absent.

At every time step of the simulation, the tracking sensors take a reading of the coordinates of the current location. As it is quite impossible to mimic the optical tracking process, wheel odometry is used as a substitute method for localisation. Since non-slip conditions are assumed in the simulation, the use of this alternative technique is deemed justified. Using this approach, the positional data is generated with the help of forward kinematics. Essentially, the current coordinates are calculated by integrating the velocities of the driving wheels within a time step to determine the distance travelled by each wheel. However, to simulate the dual-sensor optical tracking system as realistically as possible, the WMR's heading is not directly determined from wheel odometry, but from the relationship between the positional data from both sensors.

By calculating the distance between the current position of the WMR and the next checkpoint, a course can be charted along with the appropriate velocity. This completes one simulation cycle. Additionally, several PID (proportional-integral-differential) controllers are used throughout the system to fine-tune the overall performance.

4.2 Assumptions and Limitations of the Simulation Model

The general assumptions and limitations in the mathematical model apply here as well. Also, the tracking data generated by the use of forward kinematics are assumed to be a reasonably accurate simulation of actual data from the tracking sensors. In addition, the simulated tracking data assume that the WMR is operating within frictional limits at all times, and hence no slippage has occurred.

4.3 Design Details of Simulation Model

Computer modelling usually involves the feeding of data into a set of mathematical equations with the results collected afterwards. This process is repeated over a certain number of cycles and the output is compared with the predicted outcome.

The development of a simulation model using mathematical equations is generally a straightforward process. However, additional algorithms are often required to complement the

main mathematical equations and also to handle special conditions. These essential algorithms will be explained in the following sections.

4.3.1 Trajectory Generator

The objective of the study is the design of a WMR with the ability of following a predefined path or trajectory. In this simulation, the required route can be plotted via a series of points extracted from a data file, or it can be generated from an equation or series of equations. For demonstrational purposes, the latter approach was adopted. The MATLAB code used for generating the reference path can be found in Appendix A. To synchronise the path points with simulation time steps, a clock was introduced to the system as shown in Fig. 4.2. The imposition of time constraints on the desired path results in the creation of a trajectory - as defined in earlier sections. Alternatively, a signal generator can be used to provide commonly-encountered signals such as sinusoids. For the sake of data portability, the generated path points along with their associated time steps are saved in a matrix format in a data file.

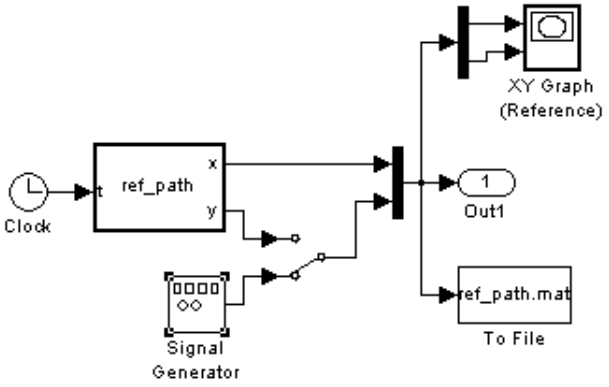


Fig. 4.2 Trajectory Generator Subsystem

4.3.2 Dual PID Controller

A PID (proportional-integral-derivative) controller is a form of feedback control device used for tuning the response of a system. It works by analysing the difference between the current and desired states of a particular process variable, and then producing a suitable response to reduce, and if possible, eliminate the error.

As can be seen in Fig. 4.1, there are two sets of dual PID controllers used in the simulation model. Each set has a pair of PID controllers because there are two parameters to manage.

The first set controls the response to the discrepancy between the present actual location and the desired position on the preset route. The positional error is split into two portions to be handled separately in relation to the x and y axes. The second set is responsible for the response to the difference between the current and required angular velocities of each wheel. The wheels are controlled independently of each other. A schematic of the dual PID controller is presented in Fig. 4.3.

The proportional parameter affects the response to the current error. It is meant for reducing rise time and overall error. While steady-state error is lessened along with the reduction of overall error, it cannot be fully eliminated. An inadequate proportional gain can cause the system to take a long time to reduce any error. This results in an unresponsive system. On the other hand, an overly aggressive setting can lead to overshoot and oscillation, and thus cause instability in the system.

The integral parameter affects the response based on the sum of errors over time. It is used for eliminating steady-state error, and will aid in pushing overall error to zero as a result. If the integral gain is too low, steady-state error will be reduced but not eliminated. However, if the setting is too high, it can result in overshoot as well as poor transient response, i.e. long settling time. At worst, it may even produce an oscillating and unstable system.

The derivative parameter affects the response to the rate of change of error. It is primarily intended for reducing overshoot and settling time. An insufficient derivative gain can result in the system being sluggish in curbing overshoot as well as ineffective in improving transient response. In contrast, an excessive setting can lead to instability due to noise amplification.

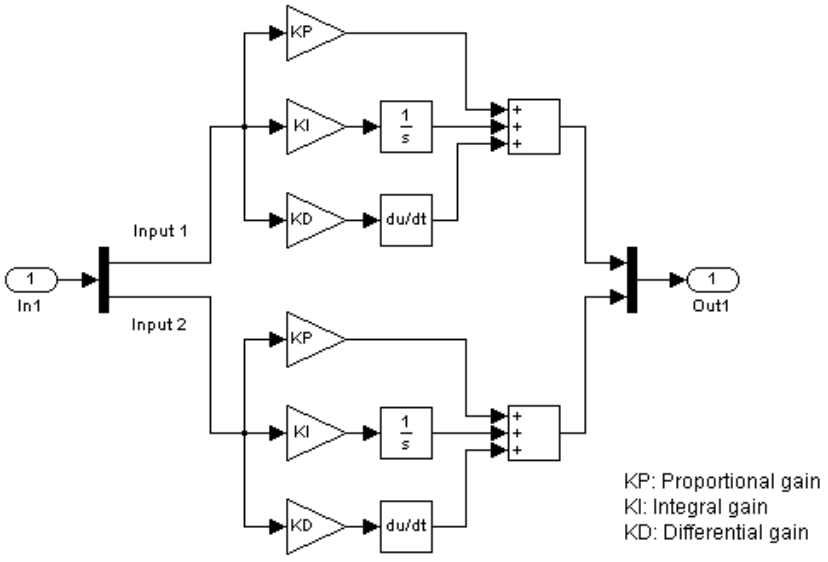


Fig. 4.3 Dual PID Controller Subsystem

4.3.3 Inverse Kinematics

In the simulation, the inverse kinematics algorithm is no more than a computational manifestation of the mathematical equations on which it is based. The process begins with the input of information on the discrepancy between the current location of the WMR and the desired position at a given point in time. This positional error is measured with respect to both the X and Y axes. The Inverse Kinematics subsystem is shown in Fig. 4.4.

In order for the WMR to reduce the positional error and get to the next destination point on the preset route within a specified timeframe, the velocity of each wheel has to be precisely controlled. This task is achieved via the use of kinematic equations described in Chapter 3.

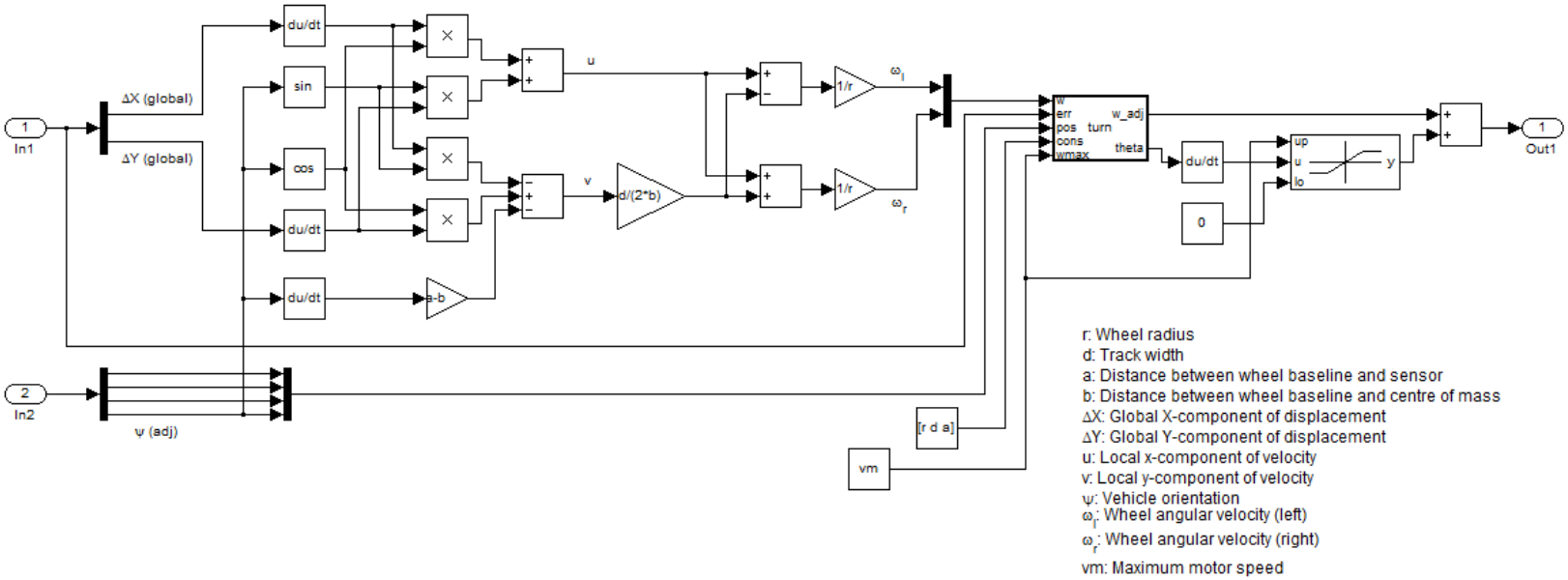


Fig. 4.4 Inverse Kinematics Subsystem

4.3.3.1 About-Turn Algorithm for Countering Reverse Motion

There are situations where the forward motion of the WMR would result in its wheels travelling a further distance to the next checkpoint than it would if the WMR moved in reverse. This condition arises when the next checkpoint is located at more than 90° clockwise or anticlockwise from the current heading of the WMR. Without any form of restriction, the control algorithm would take the shortest possible route. This could result in the WMR frequently travelling in reverse.

The simplest way to detect any potential reverse motion is to check whether the control algorithm is assigning a negative angular velocity to either of the DC motors. In order to ensure that the WMR only travels forward, the first step is to restrict the DC motors from any reverse rotation. Then, a process is initiated to turn the WMR around and drive it towards the target checkpoint.

Before proceeding on to the algorithm, the positions of key points of the WMR have to be first established. Fig. 4.5 shows the position of sensor A relative to that of the baseline centre B, as well as that of the left and right wheels L and R. From the locational data provided by sensor A, the respective positions B, L and R can be calculated geometrically. The coordinates are denoted respectively such that those for A are x_a and y_a, those for B are x_b and y_b, those for L are x_l and y_l, and those for R are x_r and y_r. The distance between A and B is a, the track width is d and the angle of orientation is ψ.

$$x_b = x_a - a\cos\psi \tag{4.1}$$

$$y_b = y_a - a\sin\psi \tag{4.2}$$

$$x_l = x_b + \frac{d}{2}\cos\left(\psi + 90^0\right) \tag{4.3}$$

$$y_l = y_b + \frac{d}{2}\sin\left(\psi + 90^0\right) \tag{4.4}$$

$$x_r = x_b + \frac{d}{2}\cos\left(\psi - 90^0\right) \tag{4.5}$$

$$y_r = y_b + \frac{d}{2}\sin\left(\psi - 90^0\right) \tag{4.6}$$

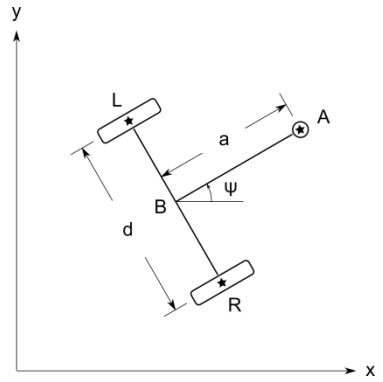


Fig. 4.5 Relative positions of WMR wheels and primary sensor

The about-turn algorithm assigns one wheel as a stationary pivot while the WMR is turned as quickly as possible until its heading is directly in line with the next checkpoint. Before this turning algorithm is applied, the angular velocities of the two motors are compared with each other, and the wheel on the side with the lower absolute velocity is ascribed as the pivot.

As can be seen from Fig. 4.6, the turning path of the WMR is basically a circular arc followed by a direct path to the next checkpoint. The points B, A, P, B_1 and Q respectively denote the centre of the wheel baseline, sensor, pivot wheel, position of the baseline centre after the WMR has turned around, and the location of the next checkpoint. Note that pivot P can be either the left or right wheel depending on the situation. In this case, P is the position of the left wheel.

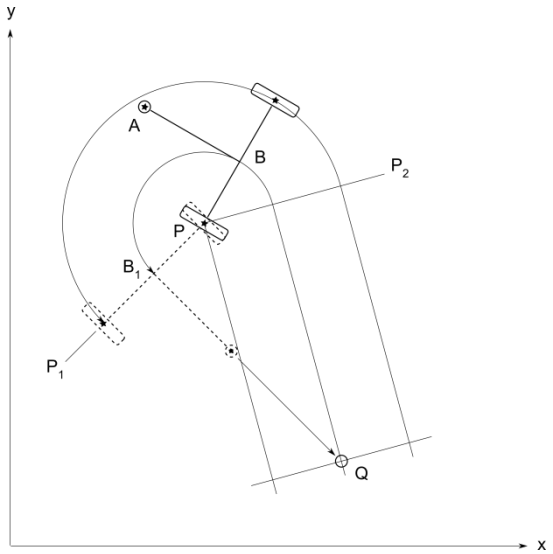


Fig. 4.6 About-turn manoeuvre of the WMR

Two tangents can be drawn from the next checkpoint to the circular path formed by the centre of the wheel baseline. One of these tangents is the path to the next checkpoint after the WMR has turned around. The normals from the pivot wheel to the two tangents are P-P_1 and P-P_2 respectively. The point B_1 is at the intersection of the tangent path and its corresponding normal.

Let r_s be the radius of the circular path formed by the centre of the wheel baseline. Pivot wheel P has coordinates x_p and y_p. As mentioned earlier, P could be either the position of the left or right wheel depending on which one is selected as the pivot. The equation for this path is:

$$\left(x - x_p\right)^2 + \left(y - y_p\right)^2 = {r_s}^2 \tag{4.7}$$

The tangent to the circular path is simply its derivative with respect to x:

$$\frac{dy}{dx} = -\frac{x - x_p}{y - y_p} \tag{4.8}$$

The coordinates of intersection point B_1 are x_1 and y_1. The parametric equations describing B_1 are thus:

$$x_1 = t \tag{4.9}$$

$$y_1 = y_p + \sqrt{{r_s}^2 - \left(t - x_p\right)^2} \tag{4.10}$$

The equation of the tangent path after the about-turn is:

$$y - y_1 = \frac{dy}{dx}\left(x - x_1\right) \tag{4.11}$$

Resolving Eqs (4.8) to (4.11),

$$\begin{aligned} &\left[\left(x - x_p\right)^2 + \left(y - y_p\right)^2\right]t^2 + \ldots \\ &\left[-2{r_s}^2\left(x - x_p\right) - 2x_p\left(x - x_p\right)^2 - 2x_p\left(y - y_p\right)^2\right]t + \ldots \\ &{x_p}^2\left(x - x_p\right)^2 + {x_p}^2\left(y - y_p\right)^2 + 2{r_s}^2 x_p\left(x - x_p\right) - {r_s}^2\left(y - y_p\right)^2 + {r_s}^4 = 0 \end{aligned} \tag{4.12}$$

To solve for t, MATLAB is used to find the roots. This quadratic equation will produce two sets of results. If the solution is complex, only the real part is considered. The longer of the chord

lengths between point B and each set of solutions will determine the position of point B_1. The chord length is defined as:

$$s = \sqrt{(x_1 - x_b)^2 + (y_1 - y_b)^2} \qquad (4.13)$$

When point B_1 has been determined, the normal vectors P-P_1 and P-P_2 can be found. Following on, the angle θ_1 between the two vectors can be established as shown in Fig. 4.7. A normal vector P-P_3 is then created to bisect the angle formed between P-P_1 and P-P_2. This bisected angle is denoted as θ_2. Next, the vector P-P_4 is projected along the wheel baseline. The angle θ_3 between P-P_3 and P-P_4 can then be calculated. Finally, θ_4, the total angular displacement required to turn the WMR around can be computed by summing angles θ_2 and θ_3.

$$\Delta\psi = \theta_4 = \theta_2 + \theta_3 \qquad (4.14)$$

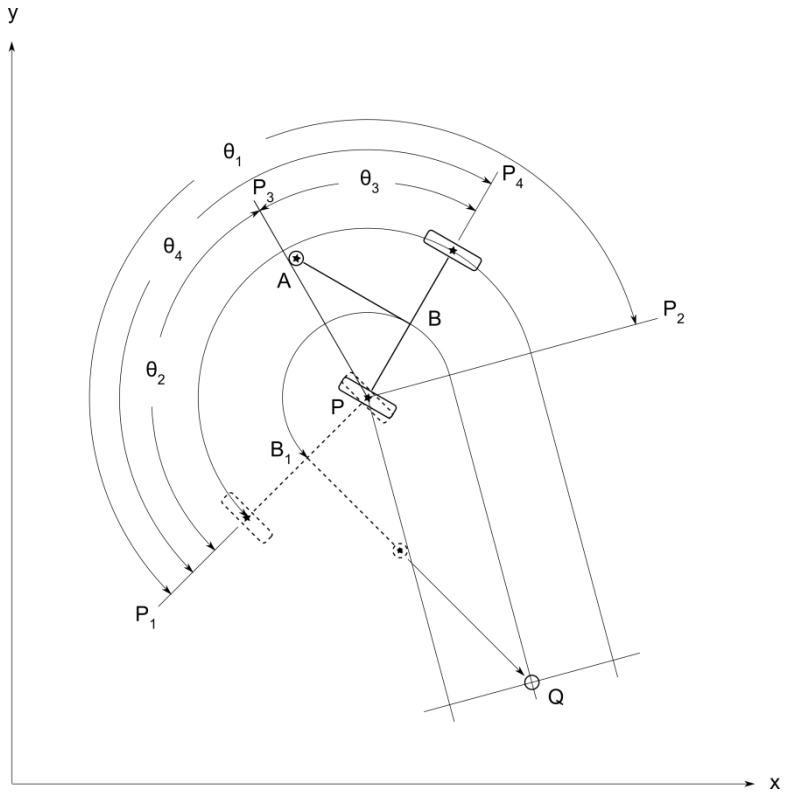


Fig. 4.7 Angular displacement of the about-turn manoeuvre

The angular velocity of the non-pivoting wheel can now be calculated to turn the WMR as quickly as possible in a circular arc spanning an angle of θ_4. Once this step is complete, the WMR only needs to follow a straight-line path to the next checkpoint.

The About-Turn algorithm is programmed as an embedded MATLAB function within the Inverse Kinematics subsystem in the Simulink model shown in Fig. 4.4. The details of the corresponding MATLAB code for the turn function block are located in Appendix A.

4.3.4 DC Motors

The locomotion of the WMR depends on a pair of DC motors independently driving a wheel each. Underpinning the simulation of the DC motors are the equations found in Chapter 3. The load torque is an estimation of the turning resistance due to factors such as Coulomb friction. With properties of the motors already established, the angular velocity of each motor can be easily determined for any given input voltage. The DC Motors subsystem is illustrated in Fig. 4.8.

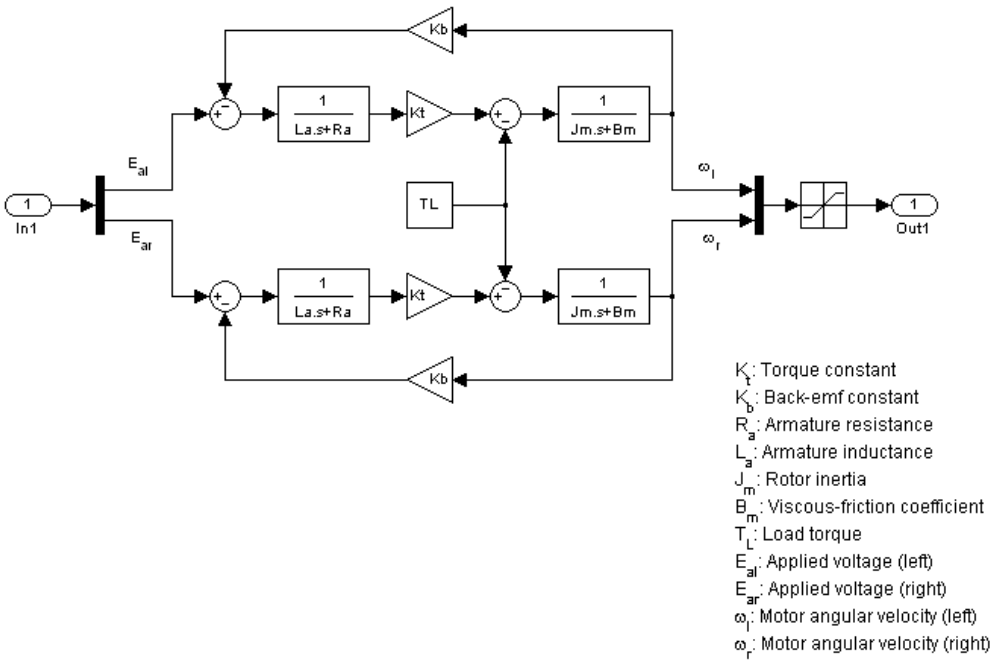


Fig. 4.8 DC Motors Subsystem

4.3.5 Motor Controller

The speed of each driving motor of the WMR varies according to situational requirements. For this purpose, a motor controller is used to regulate the speed of each motor independently. Having known the motor properties, the required voltage for each motor can be ascertained for any desired speed. In the simulation, the underlying equations for the algorithm of the motor controller are simply the reverse arrangement of the equations that describe the behaviour of the DC motor. Fig. 4.9 shows the Motor Controller subsystem.

For further elaboration, the DC Motor Modelling section of Chapter 3 is revisited. Eq (3.18) can be rearranged as:

$$J_m \frac{d^2\theta_m(t)}{dt^2} + B_m \frac{d\theta_m(t)}{dt} = K_t\, i_a(t) - T_L(t)$$

$$i_a(t) = \frac{1}{K_t}\left[J_m \frac{d^2\theta_m(t)}{dt^2} + B_m \frac{d\theta_m(t)}{dt} + T_L(t) \right] \qquad (4.15)$$

In the same manner, Eq (3.17) can be rewritten as follows:

$$L_a \frac{di_a(t)}{dt} + R_a\, i_a(t) = E_a(t) - K_b \frac{d\theta_m(t)}{dt}$$

$$E_a(t) = L_a \frac{di_a(t)}{dt} + R_a\, i_a(t) + K_b \frac{d\theta_m(t)}{dt} \qquad (4.16)$$

Remembering that the motor velocity is:

$$\omega_i = \frac{d\theta_m(t)}{dt} \qquad (where \quad i = l \text{ or } r) \qquad (4.17)$$

Thus, the necessary voltage for attaining any desired motor speed is simply obtained via the use of Eqs (4.15) to (4.17).

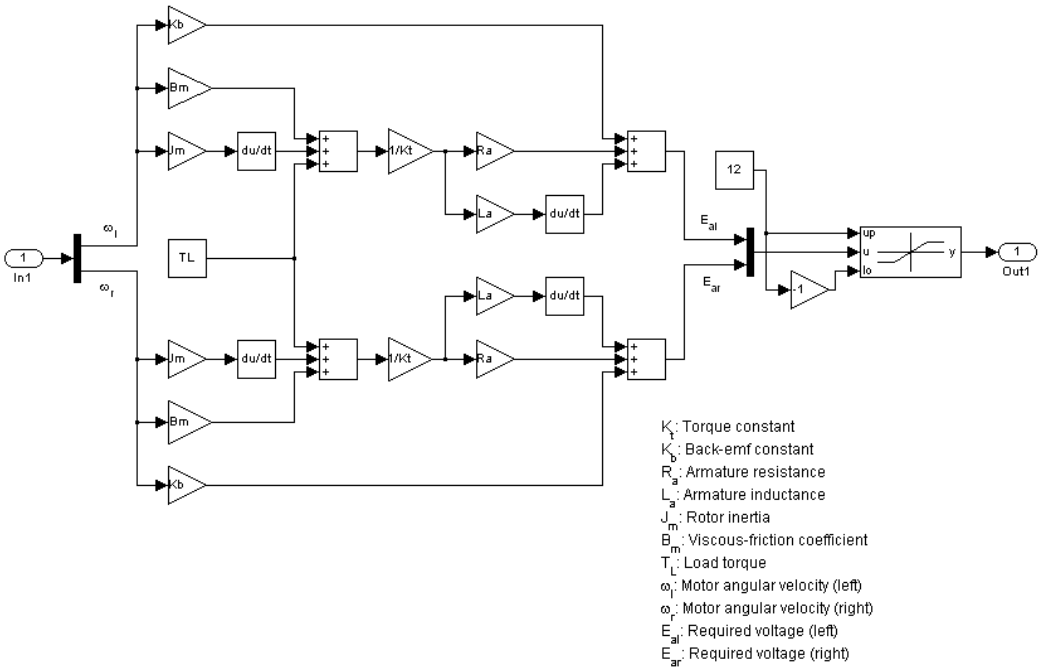


Fig. 4.9 Motor Controller Subsystem

4.3.6 Inverse Dynamics

The control system of a WMR is based fundamentally on a kinematic model. However, as explained in earlier chapters, a purely kinematic system has significant inadequacies. In order to obtain more realistic results, dynamic considerations are necessary in defining the physical limits of the kinematic model.

In the simulation, the purpose of the inverse dynamics algorithm is to constantly monitor the WMR's movement for any potential breach of dynamic boundaries that could result in wheel slippage. If a positive detection is made, the overall velocity of the WMR is reduced until it no longer exceeds the dynamic threshold.

The algorithm determines which of the wheels is likely to be responsible for the largest contribution to the overall error and proceeds to slow it down to the allowed limit. The velocity of the other wheel is reduced concurrently by the same proportion. The preservation of the velocity ratio between the wheels will ensure that the WMR maintains its heading even if its overall velocity is decreased. Ultimately, the reduction in the speed of both wheels is brought about by lowering the motor voltage by the same factor.

The Inverse Dynamics subsystem in the Simulink model has two components as shown in Fig. 4.10. The first module is a simulation of the behaviour of the pair of DC motors in response to

a voltage input that is regulated by a kinematic control system. Hence, the design of the module is identical to that of the actual DC motors subsystem. The second module consists of the inverse dynamics algorithm which calculates the predicted motion of the WMR due to motor (or wheel) speed values from the previous module. It also computes the necessary speed reduction factor if dynamic constraints are deemed to have been exceeded. The equations used for the task are found in the Dynamic Limits section of Chapter 3. The Simulink subsystem for the inverse dynamics algorithm is shown in Fig. 4.11. The inverse dynamics subsystem includes a switch that allows dynamic constraints to be bypassed, thus leaving a purely kinematic system.

The process for this portion of the general model may initially seem unnecessarily convoluted. A more intuitive and straightforward design would be to place a simple subsystem consisting of just the inverse dynamics algorithm right after the subsystem for inverse kinematics. After all, the objective of the inverse dynamics subsystem is to analyse the potential behaviour of the WMR resulting from the motor (or wheel) speed values put out by the inverse kinematics subsystem. However, as the immediate response of the DC motor to any voltage change is not linear, the resulting behaviour of the WMR is likewise affected. Hence, the current design was adopted to take into account the non-linear response of the DC motor.

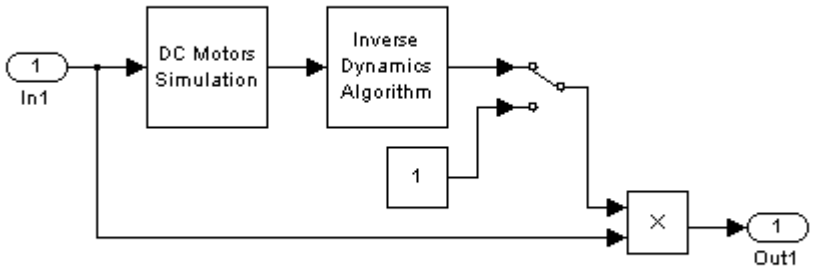


Fig. 4.10 Inverse Dynamics Subsystem

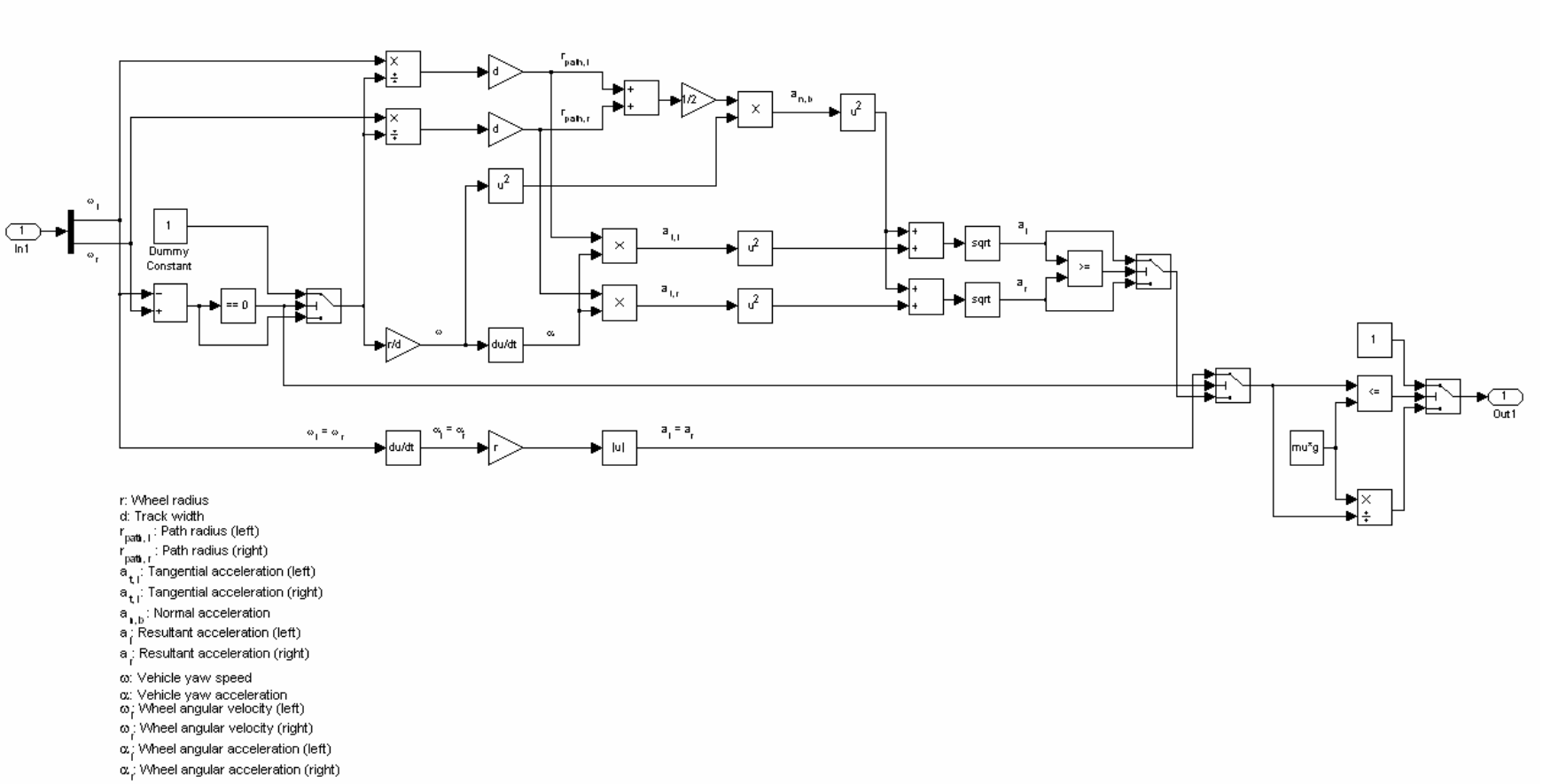


Fig. 4.11 Inverse Dynamics Algorithm Subsystem

4.3.7 Forward Kinematics (Sensor)

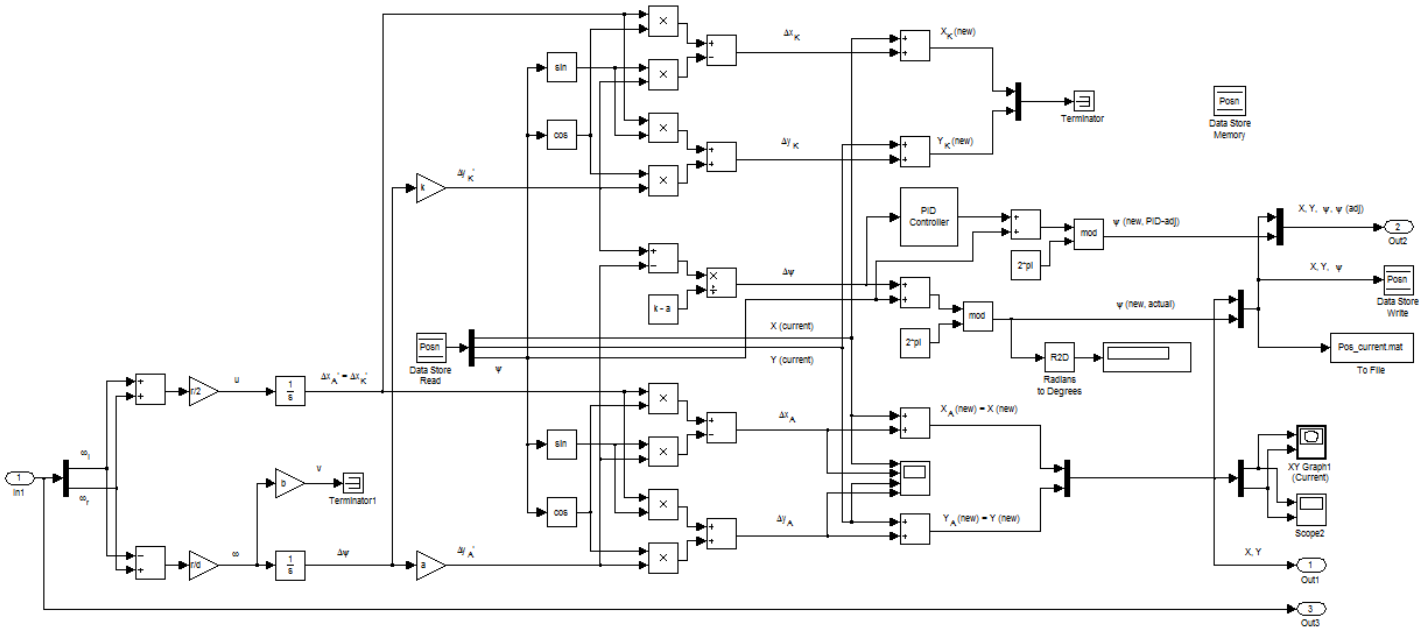
The Forward Kinematics subsystem is a simulated analogue for the optical tracking sensor. Since it is not possible to directly simulate positional data from an optical sensor, an odometric method via the use of forward kinematics is adopted instead. This approach is justified since both the optical and simulated tracking systems employ similar dead reckoning techniques. However, it is important to note that this method is only applicable in non-slip conditions.

As the name of the forward kinematics subsystem suggests, the equations used within are just a reverse arrangement of those found in the inverse kinematics subsystem. Likewise, the kinematic equations used here are also described in the same section of Chapter 3. In the simulation, the angular velocities of the wheels driven by the DC motors are used to determine the direction and distance travelled by the WMR over a specified time period.

The content within the simulation subsystem is illustrated in Fig. 4.12. The positional and orientation data are faithfully recorded. At the same time, the vehicle yaw that has been tweaked by the PID controller is sent back to the feedback loop. The single PID controller used here is of a standard type.

4.4 Summary

A computational model has been constructed by using the mathematical blueprint proposed in the previous chapter. It is comprised of both kinematic and dynamic aspects and has incorporated salient components such as PID controllers, motors and sensors. It has also taken into consideration the effects of tyre friction. With this stage completed, the behaviour of the wheeled robot in response to various assigned trajectories can now be simulated in software. The advantage of using software simulation is that trajectories and other physical parameters can be altered with ease in order to gauge the reaction of the robot. The results of the simulation will be presented in the next chapter.



r: Wheel radius
d: Track width
b: Distance between wheel baseline and centre of mass at Point G
a: Distance between wheel baseline and Sensor 1 at Point A
k: Distance between wheel baseline and Sensor 2 at Point K
$\Delta x_A'$: Displacement in local x'-coordinate with respect to Sensor 1 at Point A
$\Delta y_A'$: Displacement in local y'-coordinate with respect to Sensor 1 at Point A
$\Delta x_K'$: Displacement in local x'-coordinate with respect to Sensor 2 at Point K
$\Delta y_K'$: Displacement in local y'-coordinate with respect to Sensor 2 at Point K
Δx_A: Displacement in global x-coordinate with respect to Sensor 1 at Point A
Δy_A: Displacement in global y-coordinate with respect to Sensor 1 at Point A
Δx_K: Displacement in global x-coordinate with respect to Sensor 2 at Point K
Δy_K: Displacement in global y-coordinate with respect to Sensor 2 at Point K
x_A: Position in global x-coordinate with respect to Sensor 1 at Point A (also position of WMR)
y_A: Position in global y-coordinate with respect to Sensor 1 at Point A (also position of WMR)
x_K: Position in global x-coordinate with respect to Sensor 2 at Point K
y_K: Position in global y-coordinate with respect to Sensor 2 at Point K
ψ: Vehicle orientation
Δψ: Vehicle yaw
ω: Vehicle yaw rate
u: Velocity in local x'-coordinate respect to centre of gravity at Point G
v: Velocity in local y'-coordinate respect to centre of gravity at Point G
ω_l: Wheel angular velocity (left)
ω_r: Wheel angular velocity (right)

Fig. 4.12 Forward Kinematics (Sensor) Subsystem

Chapter 5: Simulation Results of the PID-Based Model

The purpose of the computational model developed in the preceding chapter is to simulate the behaviour of a theoretical WMR with certain physical characteristics in response to various trajectory stimuli. In this chapter, several different trajectories will be used to evaluate the performance of the model. The kinematic and dynamic versions of the model will also be tested and compared. The results from the simulation tests will provide a good indication of the validity of the model as well as the effectiveness of the PID controller.

5.1 Results and Analysis

The WMR model was tested in several simulated scenarios of which three initial basic trajectories are shown here labelled Figs. 5.1 to 5.3. In each of these simulations, the WMR was able to quickly propel itself from its starting point towards to the pre-defined path located at a certain distance away. Once the WMR has negotiated itself onto the path, it will follow it faithfully until the end of the simulation period.

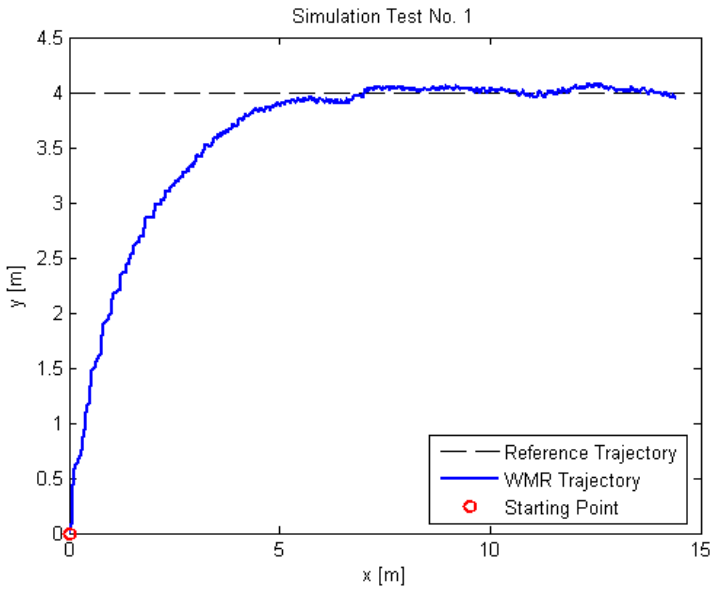


Fig. 5.1 Simulation Test No. 1: Straight trajectory with offset starting point

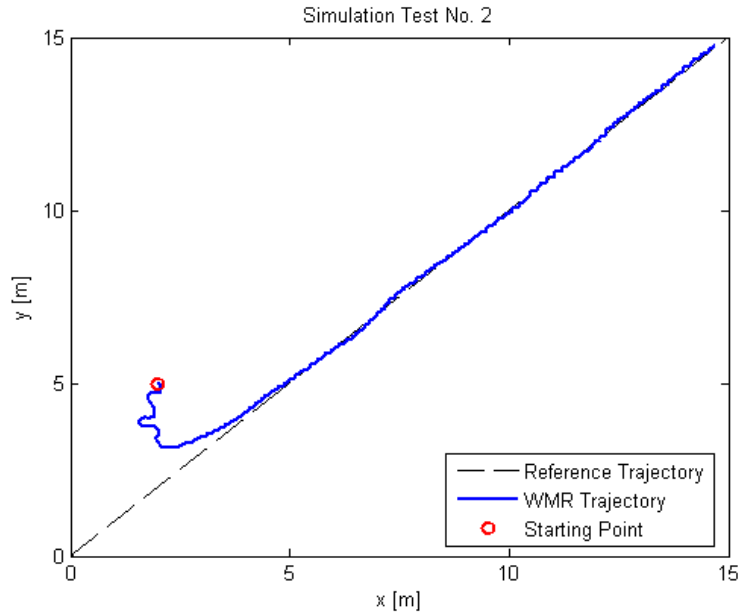


Fig. 5.2 Simulation Test No. 2: Straight trajectory with offset starting point

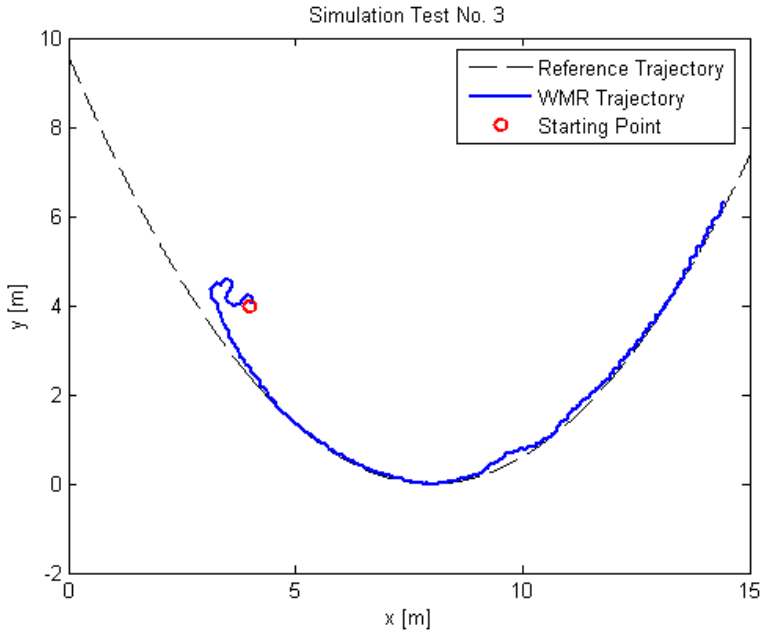


Fig. 5.3 Simulation Test No. 3: Sinusoidal trajectory with offset starting point

An important task in this study is to verify how well the kinematic and dynamic models perform against each other. For this undertaking, operational conditions must be chosen such that a clear contrast can be seen between the routes taken by both simulation models. A winding path was deemed to be most suitable because it requires the WMR to constantly vary its speed. Ultimately, the path chosen was a sinusoid as shown in Fig. 5.4.

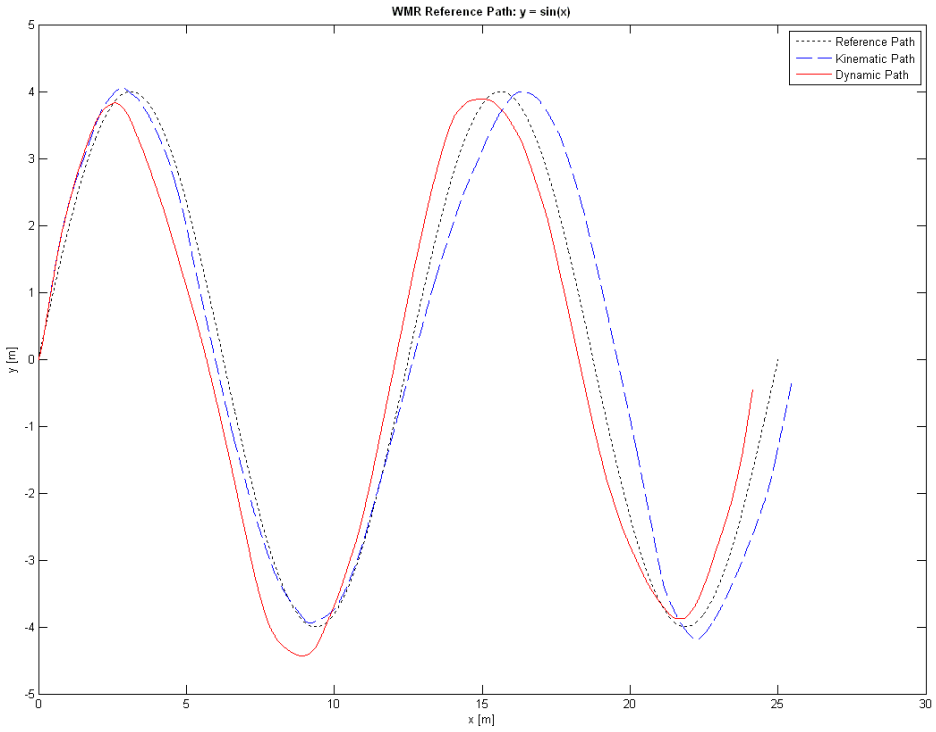


Fig. 5.4 Contrast of reference, kinematic and dynamic paths

The above plot shows three complete paths but without any information on how they progressed through time. As the dynamic model is expected to slow down at a turn more than the kinematic one, a significant difference in speed would be expected between the two models at such a location. For this reason, a few parts of the plot were selected to be redrawn for clarity as illustrated in Fig. 5.5. These segments correspond to the timeframes for the periods just before and right after the turns.

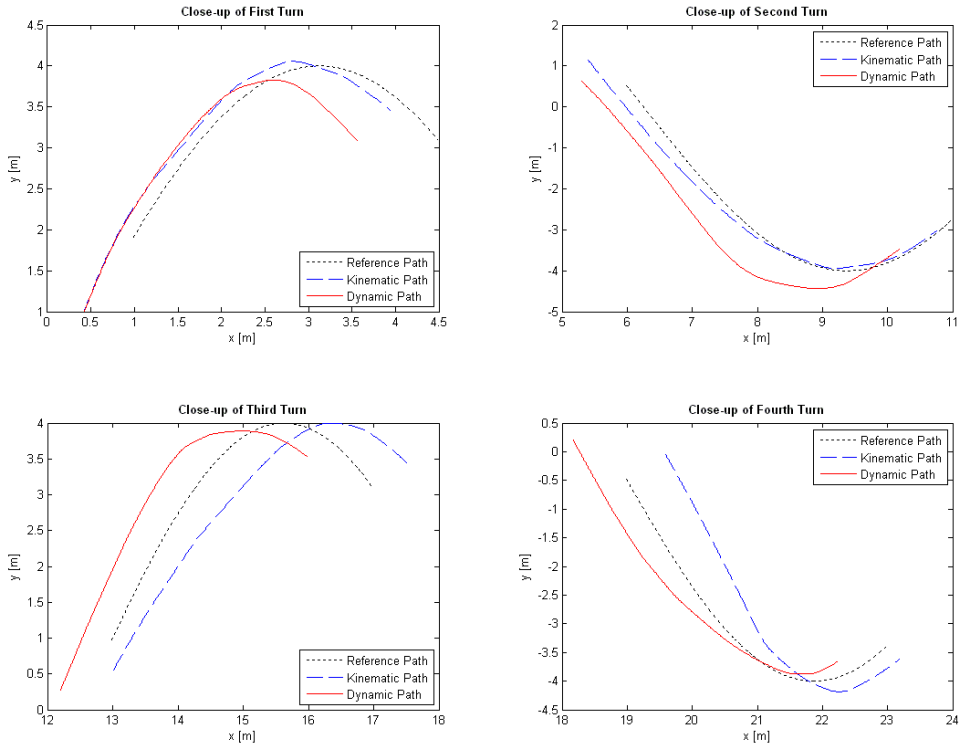


Fig. 5.5 Details of turning paths at selected timeframes

It is interesting to note from Fig. 5.5 that the path progression of the dynamic model consistently lags that of the kinematic model. It is evident that a slow-down occurs at every turn. Yet, the final outcome shows that the dynamic model is able to catch up with the kinematic route advancement. This must indicate that the dynamic model speeds up during the relatively straight segments of the course.

It is important not only for a controller to be able to follow a prescribed trajectory, but it has to be able do it with a level of accuracy that is within acceptable limits. In order to verify the tracking capabilities of the PID controller, further simulation runs were conducted. The trajectories include a range of speeds as well as turns of varying acuteness. For these tests, the gain settings of the PID controller are tuned with much more precision than the previous runs. This is to ensure that maximum attainable performance is measured and that errors are unlikely to be due to sloppy tuning. An error analysis is conducted after each run to quantify the results. The performance of the PID controller over the same course shall be compared to that of an adaptive controller in a later chapter.

There are a number of methods for judging tracking errors. Among them, the most commonly used ones are mean absolute error (MAE), mean absolute percentage error (MAPE), mean squared error (MSE) and root mean squared error (RMSE). The MSE and RMSE are considered unbiased estimators of variance (σ^2) and standard deviation (σ) respectively. That

means the MSE and RMSE are equivalent respectively to the variance and standard deviation of a sample population if it exhibits no bias. This thesis shall adopt RMS error as the main analytical metric for tracking error.

For a tracking variable x in a sample size of N, RMS error is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2} \tag{5.1}$$

$$where \quad \bar{x}\ is\ the\ desired\ value \quad and \quad tracking\ error = x - \bar{x}$$

It should be noted that the desired position of the WMR is always one time step ahead of the current position. Therefore, the tracking error is defined as the discrepancy between the current position of the WMR and the position of prescribed trajectory in the previous time step.

Thus, the tracking errors in the X- and Y-axes are respectively:

$$\varepsilon_x(t) = x(t) - x_d(t-1) \qquad where \quad x_d\ is\ the\ desired\ value \tag{5.2}$$

$$\varepsilon_y(t) = y(t) - y_d(t-1) \qquad where \quad y_d\ is\ the\ desired\ value \tag{5.3}$$

The resulting RMS error equations are:

$$RMSE(x) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\varepsilon_{x,i}{}^2} \tag{5.4}$$

$$RMSE(y) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\varepsilon_{y,i}{}^2} \tag{5.5}$$

In addition to calculating the RMS error, it is also informative to determine the error distribution and probability density function in order to be aid the visualisation of the error spread. The error PDF is plotted with the use of kernel density estimation.

The first result of the second series of simulation runs is shown Fig. 5.6. The course included three 90-degree turns, so the WMR has to slow considerably at these points and then accelerate afterwards in order to catch up to the prescribed trajectory. This path is same as the one recommended in the UMBmark calibration test (Borenstein and Feng 1995). However, it was not used here for calibration purposes because the WMR used in this research is calibrated very differently. Nevertheless, its tracking performance in this test could still be

used to compare with the results from other research. Judging from the outcome, the controller has successfully managed to follow the prescribed path.

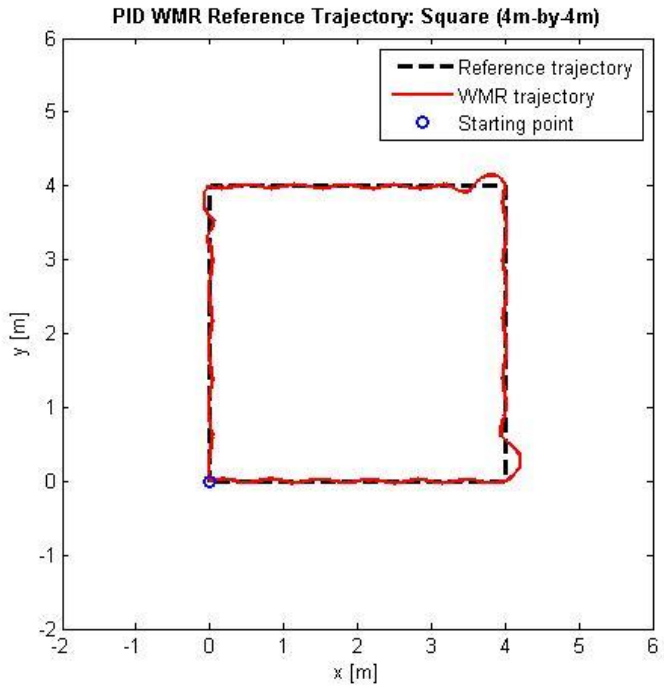


Fig. 5.6 PID Simulation Test No. 4: Square trajectory

The distribution and probability density function (PDF) of the tracking error are shown in Fig. 5.7. The results confirm that the WMR largely stayed on course and did not deviate much throughout the test. This fact was already visually evident in Fig. 5.6. The resulting RMS errors are 33.33 mm in the X-axis and 24.69 mm in the Y-axis. For a course that consisted of only straight lines, the performance was not spectacular but certainly quite acceptable. Since the controller was forbidden to engage in reverse motion for any of the wheels (for reasons described in a previous section), the turning radius of the WMR was increased. This was especially noticeable when negotiating sharp turns. As a result, the WMR veered quite a bit from the expected path while turning 90-degree corners. This behaviour may have had a significant impact on the calculation the RMS errors.

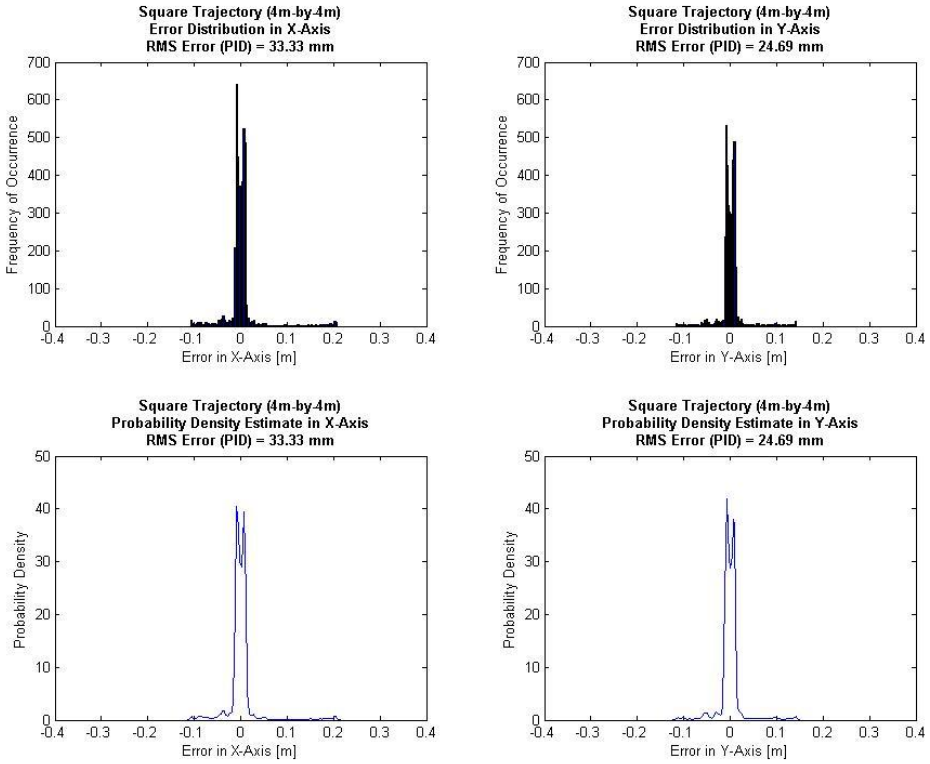


Fig. 5.7 PID Simulation Test No. 4: Square trajectory error analysis

The simulation shown in Fig. 5.8 was run with the same course as the previous test. The only difference was that the WMR and trajectory had different starting positions. This meant that right at the start, the WMR was already lagging the trajectory and had to speed up in order to catch up. With this change of circumstance, the gain settings had to be re-tuned. Based on the results, it can be concluded that the controller managed to follow the prescribed path successfully.

The lag offset starting point tested the ability of the controller to catch up to the desired trajectory in the least amount of time, but doing so without being overly aggressive to the point where the WMR would struggle to settle on the actual path when it finally got there. As the course included three 90-degree turns, the WMR had to slow considerably at these points. Once again, this presented a challenge to an aggressively-tuned controller. On the other hand, a controller that is too conservatively tuned may never be able to catch up to the trajectory given the discrepancy between the starting positions of the trajectory and WMR.

A balanced tuning is key to the success of a controller's performance. However, it is possible that the operational parameters may not allow an optimal result no matter how the controller is tuned. An error analysis of the tracking accuracy was not carried out for this test as the data from the initial offset would skew the results. The preceding test with the same starting point

for both the trajectory and WMR would be a more accurate reflection of the controllers tracking ability.

The outcome of this test was achieved after some painstaking tuning of the PID gain settings. It took only slight tweaks to the settings to have noticeably negative effects on the results. Varying the WMR starting point by more than 10 cm in any direction would also require a significant re-adjustment of the gain settings. While the PID controller has demonstrated its ability to handle the task, it is evident that it has significant limitations. It takes a fair amount of effort to tune the controller and its locus of applicability is rather small. Thus, its versatility is questionable.

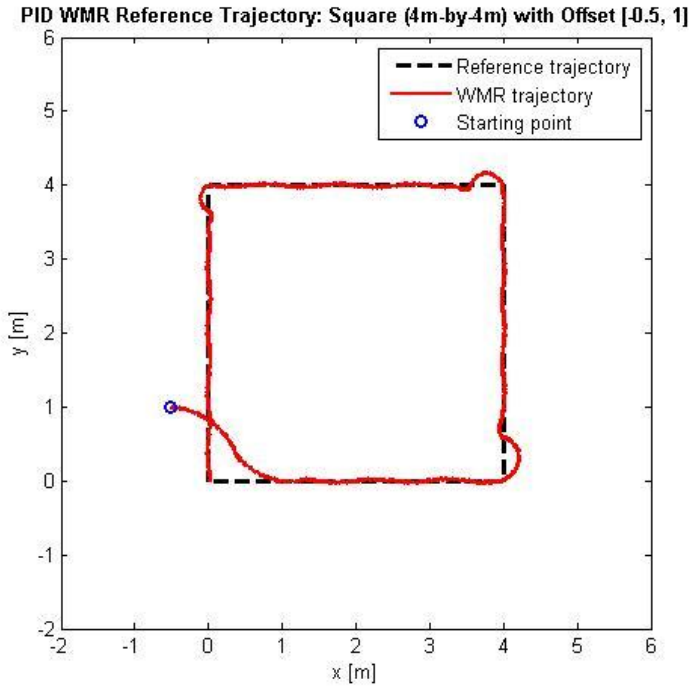


Fig. 5.8 PID Simulation Test No. 5: Square trajectory with offset starting point

At first glance, based on the trajectory plot of Fig. 5.9, the tracking performance for the sinusoidal test seemed to be superior to that of the square test. While the square trajectory had straight lines that were easier to track than the curves of a sinusoid, the latter lacked the sharp turns seen in the former. However, the error analysis as depicted in Fig. 5.10 reveals little difference between the two scenarios. The RMS errors in the X- and Y-axes are 22.76 mm and 33.30 mm respectively. So, the magnitude of the deviations in the X- and Y-axes are virtually reversed between the two tests. The error spreads of both tests are also similar. This means that the controller was able to track the sinusoidal path as well as the square one, but not distinctly better. It appears that the advantage of not having to negotiate sharp corners was negated by the increased difficulty in tracking a curved trajectory.

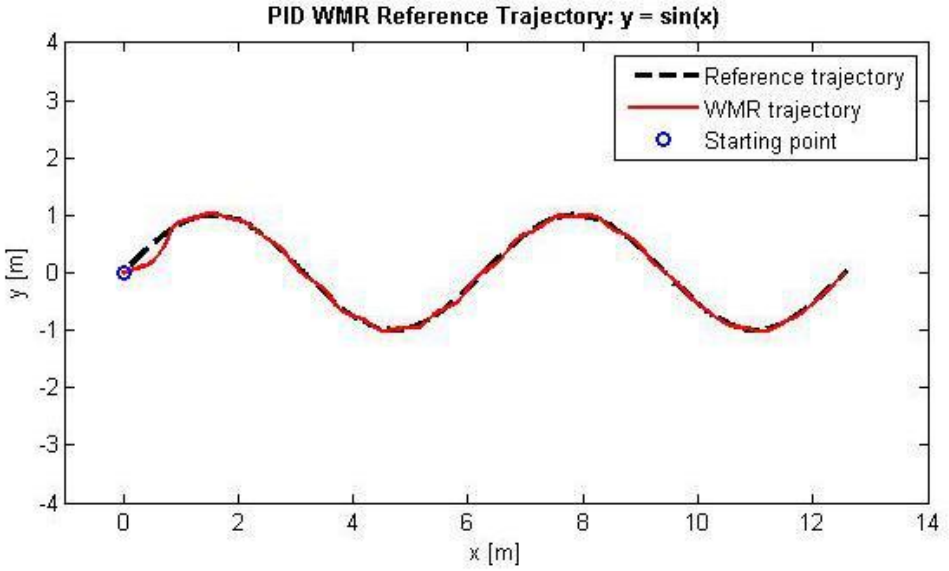


Fig. 5.9 PID Simulation Test No. 6: Sinusoidal trajectory

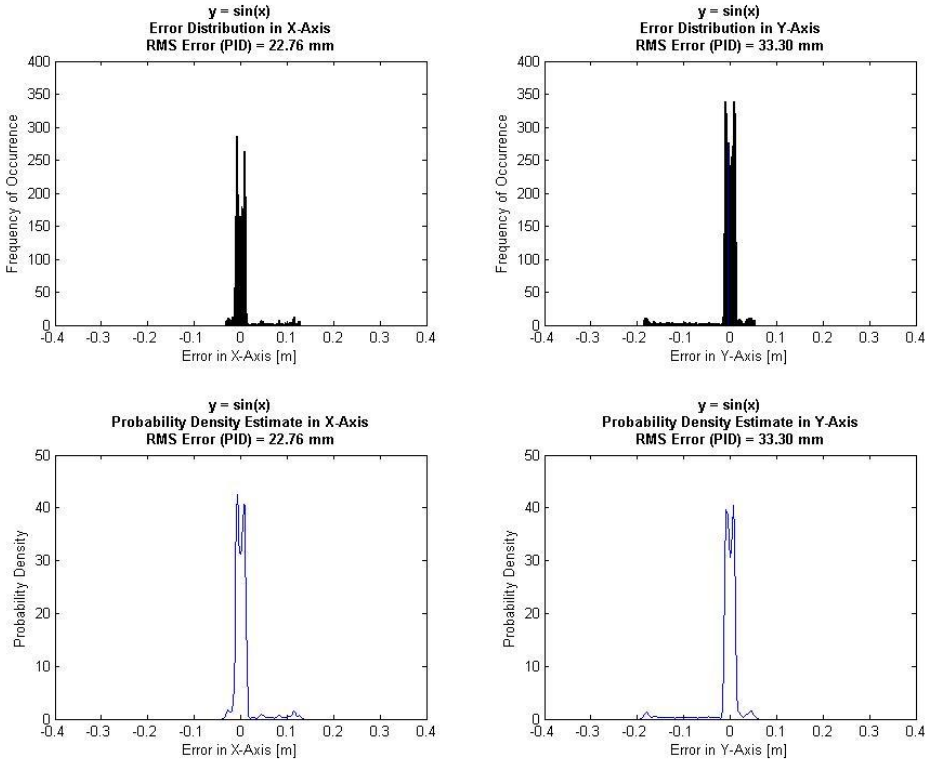


Fig. 5.10 PID Simulation Test No. 6: Sinusoidal trajectory error analysis

The sinusoidal trajectory was run again with slightly different parameters as shown in Fig. 5.11. The turns involved were more acute and the path was a little more than twice the length of previous one, and the time allotted for completion of the course was doubled. This meant that although the WMR had to slow down more drastically than previously to negotiate tighter turns, it was not given any more time to regain speed after every turn. Once again, the gain settings had to be altered substantially. Once again, the plot shows that the controller was able to accomplish its task. A closer look at the error distribution and density shown in Fig. 5.12 indicates that although the WMR had mostly followed the expected trajectory, it veered off the track more often and by a greater margin than the previous sinusoidal test. Given that the course was more demanding for reasons mentioned above, the outcome was not unexpected. The RMS errors in the X- and Y-axes are 24.62 mm and 62.94 mm respectively. The discrepancy in the X-axis is marginally greater than in the preceding test, but for Y-axis it is almost doubled. If the WMR is required to negotiate a very narrow course (that may be walled like a maze), this controller may struggle to perform with such error margins.

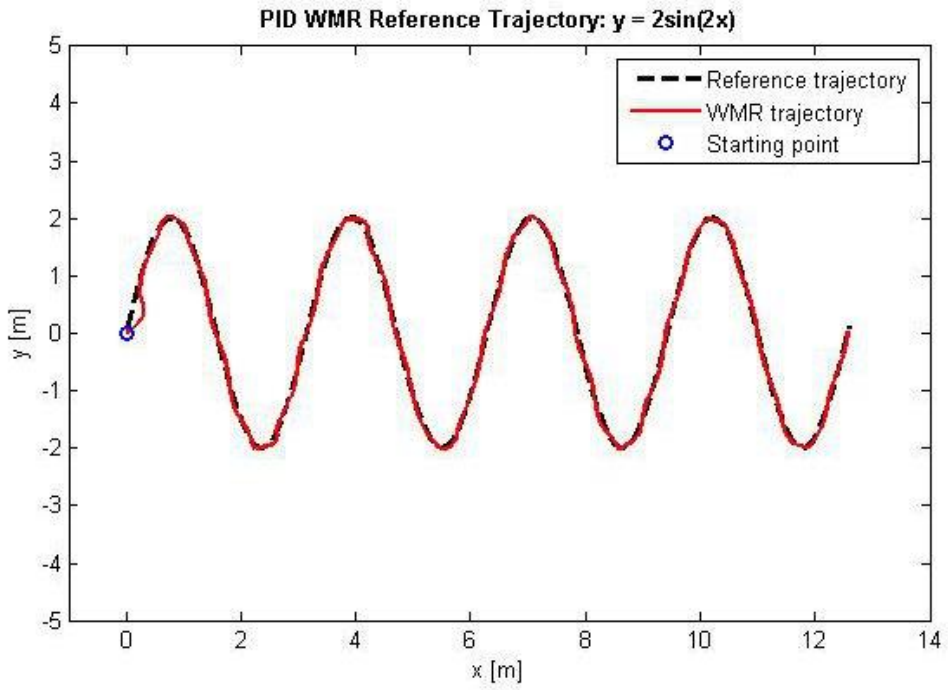


Fig. 5.11 PID Simulation Test No. 7: Sinusoidal trajectory 2

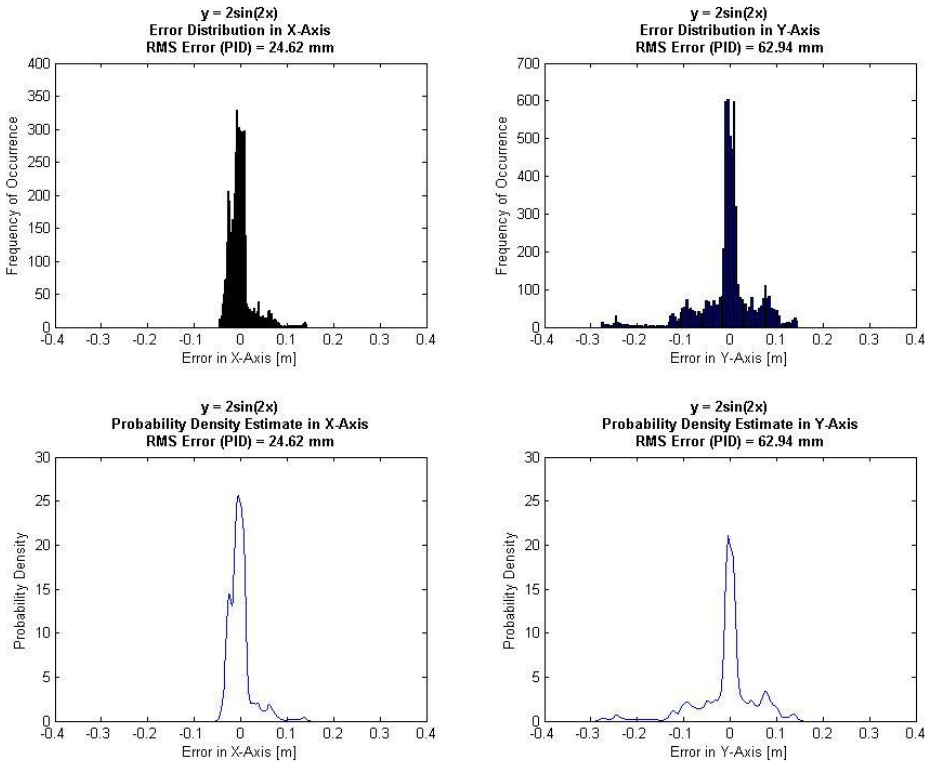


Fig. 5.12 PID Simulation Test No. 7: Sinusoidal trajectory 2 error analysis

For the last simulation run, a circular trajectory was chosen. In addition to testing tracking performance of the controller, it checked the ability of the WMR to return to its starting point - just like in the square track scenario. There were no sharp turns or sudden changes in speed or heading in this course. So, the results were expected to be quite good. As illustrated in Fig 5.13, the controller was quite up to the task. During the tuning of the PID controller's gain settings, it was noticed that the WMR would often follow the track quite faithfully until just a moment before it arrived back at the starting point. At that instance, some slight oscillations became noticeable. Extending the course time revealed that further oscillations occurred sporadically although they were quite minor. The gain settings were then adjusted such that the controller could perform reliably over two revolutions of the course. When that was achieved, the test was performed for even long distances, i.e. up to 10 revolutions. It was confirmed for this test that as long as the controller could function predictably over two revolutions, the same gain settings would apply for much longer runs. So, the test was re-run to cover two revolutions of the same track just to see how well the WMR could hold its course over a greater distance. Since an overlapping plot had a tendency to obscure undulations and minor details of the WMR's path, the graph for a single revolution of the track is included here (Fig. 5.13) for visual clarity.

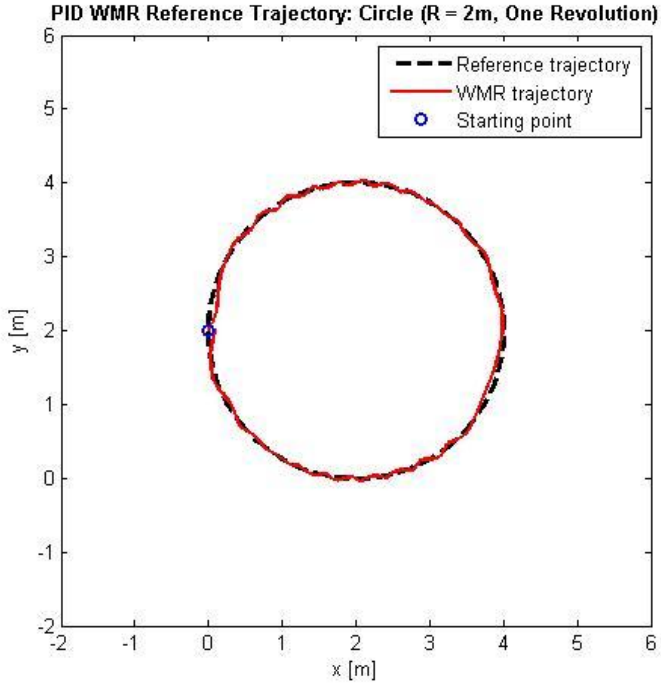


Fig. 5.13 PID Simulation Test No. 8: Circular trajectory (1 rev)

It is clear from Fig. 5.14 that the controller was able to keep the WMR on track for at least two revolutions of the circular trajectory. While the error spread shown in Fig. 5.15 suggests that the WMR closely followed the path most of the time, there are times when it did not track its trajectory well at all. This could mean that it either strayed from the course by quite a significant margin or it maintained its course but lagged behind the desired position at certain points. Judging from Fig. 5.14, the latter is the more likely explanation. The resulting error analysis does not point to a good controller performance. While the RMS error in the X-axis is a respectable 21.85 mm, the discrepancy in the Y-axis is 134.65 mm. This last result is rather poor considering that the prescribed trajectory was not very challenging. Many further attempts were made to improve the gain settings, but a better outcome proved elusive.

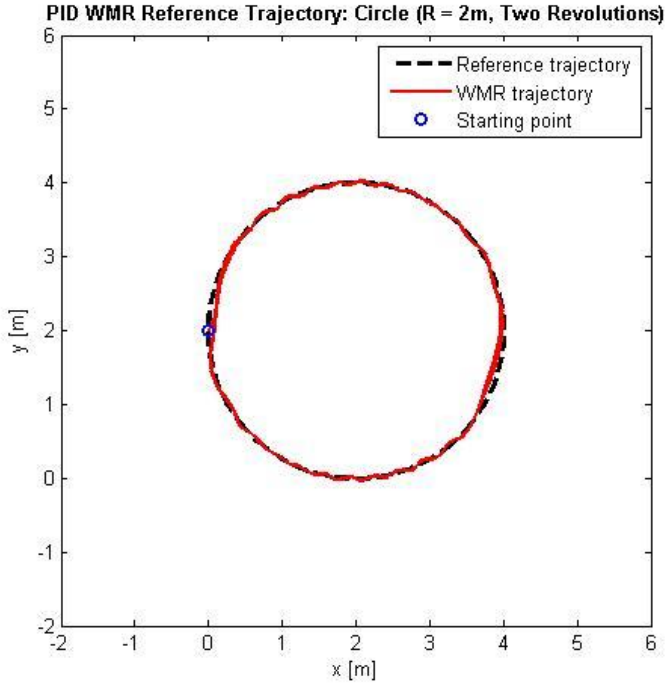


Fig. 5.14 PID Simulation Test No. 8: Circular trajectory (2 revs)

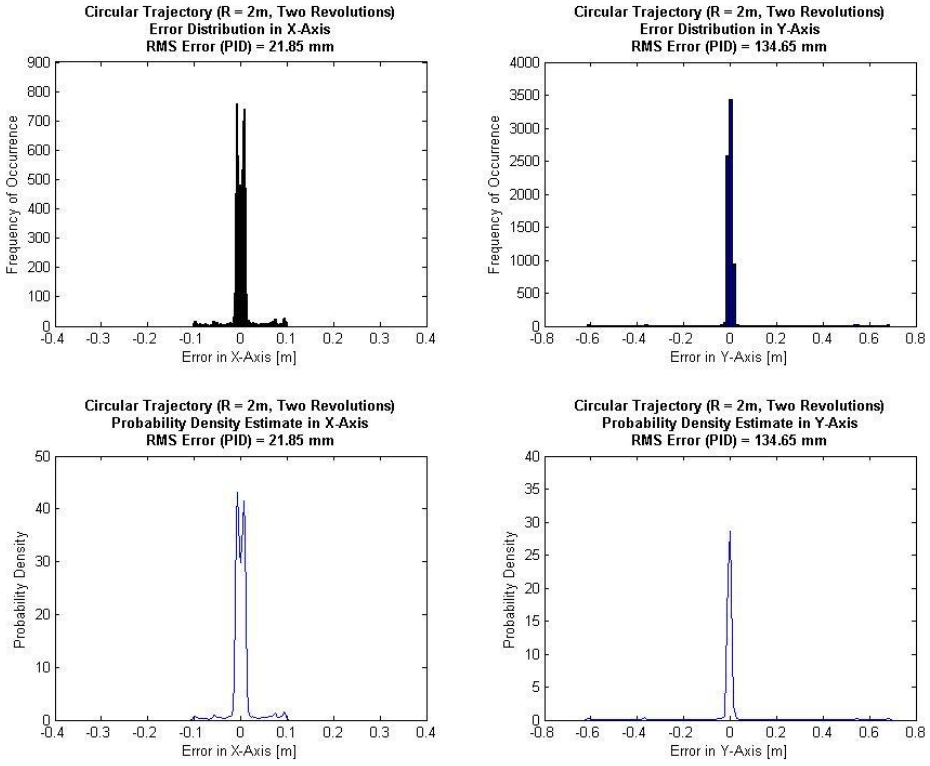


Fig. 5.15 PID Simulation Test No. 8: Circular trajectory (2 revs) error analysis

5.2 Discussion and Conclusion

The results prove that the dynamic algorithm will slow down the simulated WMR when situations arise that could cause it to exceed friction limits. Nevertheless, it is responsive enough to be able to speed up when required to match the reference trajectory. In general, the PID model performs reasonably well in a simulated environment and demonstrates the feasibility of the idea. However, it is not without shortcomings.

The PID control algorithm used in this study is relatively basic. Based on the error analyses, its tracking performance was not uniform across various situations. A summary of results is displayed in Table 5.1. In some scenarios, the controller tracks with very good accuracy while in others it struggles to stay on course. It is certainly possible that the chosen gain settings were not optimal. However, as stated before, there is no commonly-adopted method for tuning cascaded PID systems, and the applicability of PID control to a highly non-linear system is questionable. So, manual tuning had to be relied upon to obtain the most suitable system response for any given condition.

Trajectory Type	RMS Error in X-Axis (mm)	RMS Error in Y-Axis (mm)
Square: 4m x 4m	33.33	24.69
Sinusoidal: y = sin(x)	22.76	33.30
Sinusoidal: y = 2sin(2x)	24.62	62.94
Circular: radius = 2m	21.85	134.65

Table 5.1 Summary of simulation test results of PID model

It was discovered during the simulation that the gain settings applied to only a small locus of operational parameters. This means that every time there is a minor change in the scenario, such as the desired trajectory, required average speed or different starting offset, a re-tune of the gain settings would be needed. Furthermore, minor tweaks to the settings often seemed to have noticeable effects on the final outcome. So, the adjustments had to be done in very fine increments or decrements. This means that the tuning procedure requires a lot of time and effort.

Thus, not only do the gain settings of the controller require frequent adjustments, the adjustment process itself is also painstaking. These characteristics suggest that the controller in its current form has limited practical use. It would be necessary for an improved version or an entirely new controller to be designed before a robotic vehicle with dual optical sensors could be built and competently tested.

For now, it must be remembered that without an actual working prototype, how well these results match up to reality remains to be seen.

5.3 Further Work

After a review of current literature and careful consideration, it is decided that a new controller would be the best way forward. This new controller has to be able to cope with a greater range of operational situations without the need for constant adjustments. Indeed, it would be preferable if it could even tune itself while on the move. Based on these preferences, an adaptive algorithm is the natural choice for the task at hand. If and when the new controller has proved its superior capabilities in simulation, a working prototype would be constructed in order to validate the theoretical results.

Chapter 6: Analysis and Design of the Adaptive System

In the earlier chapters of this thesis, a mathematical model of the wheeled robot was analysed and developed. It provided the basis for the computational representation that was designed subsequently. While the PID-based model was able to perform adequately in simulation, it also revealed a number of inherent deficiencies that could not be overcome by introducing minor alterations. This outcome has led to the need for a major overhaul or even a new model.

In this chapter, an extensively-modified model with a completely different control system will be developed to address the problems faced by the previous model. This will be a fully-dynamic system with an adaptive controller. In addition, the unique characteristics of the optical mouse sensor will be explained, and two methods for calculating odometry will be analysed and compared. A technique for achieving partial redundancy will also be introduced.

6.1 DC Motors, Gear Assembly and Vehicular Dynamics

The dynamics of the DC motor and vehicle have already been discussed in Chapter 3. However, they are revisited here in more detail. The equations of the DC motor and gear assembly are now revised to include a gear ratio, n_g. These equations are expressed in terms of Kirchhoff's and Newton's laws as shown in the following. Note that subscripts m and L refer to variables associated with the motor and axle loads respectively.

The gear ratio is defined as:

$$n_g = \frac{\theta_m}{\theta_L} \qquad \left(where \quad n_g \geq 1\right) \tag{6.1}$$

Rewriting Eq (3.22),

$$J_m\ddot{\theta}_m(t) + B_m\dot{\theta}_m(t) + \frac{1}{n_g}\left[J_L\ddot{\theta}_L(t) + B_L\dot{\theta}_L(t) + T_L\right] = T_m$$

$$J_m\ddot{\theta}_m(t) + B_m\dot{\theta}_m(t) + \frac{1}{n_g{}^2}\left[J_L\ddot{\theta}_m(t) + B_L\dot{\theta}_m(t)\right] = T_m - \frac{T_L(t)}{n_g} \tag{6.2}$$

Or, $$n_g\left[J_m\ddot{\theta}_L(t) + B_m\dot{\theta}_L(t)\right] + \frac{1}{n_g}\left[J_L\ddot{\theta}_L(t) + B_L\dot{\theta}_L(t)\right] = T_m - \frac{T_L(t)}{n_g} \tag{6.3}$$

Substituting Eq (3.19) into Eqs (6.3) and (6.4),

$$J_m\ddot{\theta}_m(t)+B_m\dot{\theta}_m(t)+\frac{1}{n_g{}^2}\left[J_L\ddot{\theta}_m(t)+B_L\dot{\theta}_m(t)\right]=K_t i_a(t)-\frac{T_L(t)}{n_g} \tag{6.4}$$

$$n_g\left[J_m\ddot{\theta}_L(t)+B_m\dot{\theta}_L(t)\right]+\frac{1}{n_g}\left[J_L\ddot{\theta}_L(t)+B_L\dot{\theta}_L(t)\right]=K_t i_a(t)-\frac{T_L(t)}{n_g} \tag{6.5}$$

From Eqs (3.20) and (3.21),

$$L_a\frac{di_a(t)}{dt}+R_a\ i_a(t)=E(t)-K_b\frac{d\theta_m(t)}{dt} \tag{6.6}$$

Using Laplace Transforms on Eq (6.6),

$$I_a(s)=\frac{E(s)-K_b s\theta_m(s)}{L_a s+R_a} \tag{6.7}$$

Considering inertial load only (i.e. $T_L = 0$), and applying Eq (6.7) to Eq (6.4),

$$s\left[J_m s+B_m+\frac{1}{n_g{}^2}\left(J_L s+B_L\right)\right]\theta_m(s)=K_t\left[\frac{E(s)-K_b s\theta_m(s)}{L_a s+R_a}\right]$$

$$s\left(\frac{L_a s+R_a}{K_t}\right)\left[J_m s+B_m+\frac{1}{n_g{}^2}\left(J_L s+B_L\right)\right]\theta_m(s)+K_b s\theta_m(s)=E(s)$$

$$\frac{\theta_m(s)}{E(s)}=\frac{K_t}{s\left\{\left(L_a s+R_a\right)\left[J_m s+B_m+\frac{1}{n_g{}^2}\left(J_L s+B_L\right)\right]+K_t K_b\right\}} \tag{6.8}$$

Or,
$$\frac{\theta_L(s)}{E(s)}=\frac{K_t}{n_g s\left\{\left(L_a s+R_a\right)\left[J_m s+B_m+\frac{1}{n_g{}^2}\left(J_L s+B_L\right)\right]+K_t K_b\right\}} \tag{6.9}$$

Evaluating Eq (6.9) in the time domain,

$$\begin{aligned}&\left(n_g{}^2 L_a J_m+L_a J_L\right)\dddot{\theta}_L(t)+\left(n_g{}^2 L_a B_m+L_a B_L+n_g{}^2 R_a J_m+R_a J_L\right)\ddot{\theta}_L(t)\\&+\left(n_g{}^2 R_a B_m+R_a B_L+n_g{}^2 K_t K_b\right)\dot{\theta}_L(t)=n_g K_t E\end{aligned} \tag{6.10}$$

Taking Eq (6.7) this time with E = 0, and applying to Eq (6.4),

$$I_a(s) = -\frac{K_b s\theta_m(s)}{L_a s + R_a} \tag{6.11}$$

$$s\left[J_m s + B_m + \frac{1}{n_g{}^2}\left(J_L s + B_L\right)\right]\theta_m(s) = -K_t\left[\frac{K_b s\theta_m(s)}{L_a s + R_a}\right] - \frac{T_L(s)}{n_g}$$

$$s\left[J_m s + B_m + \frac{1}{n_g{}^2}\left(J_L s + B_L\right) + \frac{K_t K_b}{L_a s + R_a}\right]\theta_m(s) = -\frac{T_L(s)}{n_g}$$

$$\frac{\theta_m(s)}{T_L(s)} = -\frac{L_a s + R_a}{s\left\{\left(L_a s + R_a\right)\left[J_m s + B_m + \frac{1}{n_g{}^2}\left(J_L s + B_L\right)\right] + k_t k_b\right\}} \tag{6.12}$$

Or,
$$\frac{\theta_L(s)}{T_L(s)} = -\frac{L_a s + R_a}{n_g s\left\{\left(L_a s + R_a\right)\left[J_m s + B_m + \frac{1}{n_g{}^2}\left(J_L s + B_L\right)\right] + k_t k_b\right\}} \tag{6.13}$$

Evaluating Eq (6.13) in the time domain,

$$\begin{aligned}&\left(n_g{}^2 L_a J_m + L_a J_L\right)\dddot{\theta}_L(t) + \left(n_g{}^2 L_a B_m + L_a B_L + n_g{}^2 R_a J_m + R_a J_L\right)\ddot{\theta}_L(t)\\ &+ \left(n_g{}^2 R_a B_m + R_a B_L + n_g{}^2 K_t K_b\right)\dot{\theta}_L(t) = -\left[L_a \dot{T}_L(t) + R_a T_L(t)\right]\end{aligned} \tag{6.14}$$

Equations (6.10) and (6.14) clearly describe a third-order system. The third-order term is the derivative of the angular acceleration of the load attached to the motor. Although the term exists in the ideal mathematical model, it has little physical meaning and adds ambiguity and complexity to the development of a control system. Hence, it is advantageous to eliminate the term if possible.

In order to remove the third-order term, the following coefficients have to be ignored, but only if they are justifiably insignificant:

$$L_a J_m \quad and \quad L_a J_L$$

In general, the electrical time constant of a motor is often very small and may be ignored. It will be shown in a later chapter that this assumption holds true according to the specifications of the motors used in the prototype. Thus,

$$\frac{L_a}{R_a} \approx 0 \tag{6.15}$$

Based on this simplification, the third-order term vanishes and the equations can now be reduced to a second-order system. The validity of this simplification and the control algorithm based on these equations will later be verified by computational simulation as well as hardware testing.

Dividing Eqs (6.10) and (6.14) by R_a,

$$\left(n_g{}^2 J_m + J_L\right)\ddot{\theta}_L(t) + \left(n_g{}^2 B_m + B_L + \frac{n_g{}^2 K_t K_b}{R_a}\right)\dot{\theta}_L(t) = \frac{n_g K_t}{R_a} E(t) \quad (6.16)$$

$$\left(n_g{}^2 J_m + J_L\right)\ddot{\theta}_L(t) + \left(n_g{}^2 B_m + B_L + \frac{n_g{}^2 K_t K_b}{R_a}\right)\dot{\theta}_L(t) = -T_L(t) \quad (6.17)$$

For convenience of calculation, some parameters may be grouped together,

$$J_{eq} = n_g{}^2 J_m + J_L \quad (6.18)$$

$$B_{eq} = n_g{}^2 B_m + B_L \quad (6.19)$$

Now, Eqs (6.16) and (6.17) may be written as:

$$J_{eq}\ddot{\theta}_L(t) + \left(B_{eq} + \frac{n_g{}^2 K_t K_b}{R_a}\right)\dot{\theta}_L(t) = n_g T_m(t) \quad (6.20)$$

And $$J_{eq}\ddot{\theta}_L(t) + \left(B_{eq} + \frac{n_g{}^2 K_t K_b}{R_a}\right)\dot{\theta}_L(t) = -T_L(t) \quad (6.21)$$

Using the principle of superposition,

$$J_{eq}\ddot{\theta}_L(t) + \left(B_{eq} + \frac{n_g{}^2 K_t K_b}{R_a}\right)\dot{\theta}_L(t) = n_g T_m(t) - T_L(t) \quad (6.22)$$

Or, $$J_{eq}\ddot{\theta}_L(t) + \left(B_{eq} + \frac{n_g{}^2 K_t K_b}{R_a}\right)\dot{\theta}_L(t) = n_g \frac{K_t}{R_a} E(t) - T_L(t) \quad (6.23)$$

Let $\theta = \theta_L$, and subscript i = l or r, for left or right wheel respectively,

$$T_{L,i} = \frac{n_g K_t}{R_a} E_i - J_{eq}\ddot{\theta}_i - \left(B_{eq} + \frac{{n_g}^2 K_t K_b}{R_a} \right) \dot{\theta}_i \tag{6.24}$$

Or, $$T_{L,i} = \frac{n_g K_t}{R_a} E_i - J_{eq}\dot{\omega}_i - \left(B_{eq} + \frac{{n_g}^2 K_t K_b}{R_a} \right) \omega_i \tag{6.25}$$

Load torque on each wheel is

$$T_{L,i} = F_{x',i} \cdot r \tag{6.26}$$

Substituting Eqs (6.25) and (6.26) into (3.24),

$$\dot{u} = \frac{1}{m}\left(F_{x',l} + F_{x',r}\right) + v\omega$$

$$\dot{u} = \frac{1}{mr}\left[\left(\frac{n_g K_t}{R_a} \right) E_l - J_{eq}\dot{\omega}_l - \left(B_{eq} + \frac{{n_g}^2 K_t K_b}{R_a} \right) \omega_l + \left(\frac{n_g K_t}{R_a} \right) E_r - J_{eq}\dot{\omega}_r - \left(B_{eq} + \frac{{n_g}^2 K_t K_b}{R_a} \right) \omega_r \right] + v\omega$$

But $v = b\omega$,

$$\dot{u} = \frac{1}{mr}\left[\left(\frac{n_g K_t}{R_a} \right) \left(E_l + E_r\right) - J_{eq}\left(\dot{\omega}_l + \dot{\omega}_r\right) - \left(B_{eq} + \frac{{n_g}^2 K_t K_b}{R_a} \right) \left(\omega_l + \omega_r\right) \right] + b\omega^2 \tag{6.27}$$

From Eq (3.1),

$$u = r\,\frac{\left(\omega_l + \omega_r\right)}{2}$$

$$\omega_l + \omega_r = \frac{(2u)}{r}$$

Substituting into Eq (6.27),

$$\dot{u} = \frac{1}{mr}\left[\left(\frac{n_g K_t}{R_a} \right) \left(E_l + E_r\right) - J_{eq}\left(\frac{2\dot{u}}{r} \right) - \left(B_{eq} + \frac{{n_g}^2 K_t K_b}{R_a} \right) \left(\frac{2u}{r} \right) \right] + b\omega^2$$

$$\left(1+\frac{J_{eq}}{mr^2}\right)\dot{u}=-\frac{2}{mr^2}\left(B_{eq}+\frac{{n_g}^2K_tK_b}{R_a}\right)u+\left(\frac{n_gK_t}{R_amr}\right)\left(E_l+E_r\right)+b\omega^2$$

$$\frac{R_a}{n_gK_t}\left(mr+\frac{2J_{eq}}{r}\right)\dot{u}-\frac{2}{r}\left(\frac{R_aB_{eq}}{n_gK_t}+n_gK_b\right)u-\frac{R_amrb}{n_gK_t}\omega^2=E_l+E_r \qquad (6.28)$$

Substituting Eqs (6.25), (6.26) and (3.25) into (3.26),

$$F_{y'l}+F_{y'r}=m\left(\dot{v}+u\omega\right)$$

$$\therefore \qquad \dot{\omega}=\frac{1}{I_z}\left[\left(F_{x'l}-F_{x'r}\right)\frac{d}{2}-mb\left(b\dot{\omega}+u\omega\right)\right]$$

$$\dot{\omega}=\frac{1}{I_z}\left\{\frac{d}{2r}\left[\left(\frac{n_gK_t}{R_a}\right)E_r-J_{eq}\dot{\omega}_r-\left(B_{eq}+\frac{{n_g}^2K_tK_b}{R_a}\right)\omega_r-\left(\frac{n_gK_t}{R_a}\right)E_l+J_{eq}\dot{\omega}_l-\left(B_{eq}+\frac{{n_g}^2K_tK_b}{R_a}\right)\omega_l\right]-mb\left(b\dot{\omega}+u\omega\right)\right\}$$

$$\dot{\omega}=\frac{1}{I_z}\left\{\frac{d}{2r}\left[\left(\frac{n_gK_t}{R_a}\right)\left(E_r-E_l\right)-J_{eq}\left(\dot{\omega}_r-\dot{\omega}_l\right)-\left(B_{eq}+\frac{{n_g}^2K_tK_b}{R_a}\right)\left(\omega_r-\omega_l\right)\right]-mb\left(b\dot{\omega}+u\omega\right)\right\} \qquad (6.29)$$

From Eq (3.2),

$$v=\frac{br}{d}\left(\omega_r-\omega_l\right)$$

$$b\omega=\frac{br}{d}\left(\omega_r-\omega_l\right)$$

$$\omega_r-\omega_l=\frac{d\omega}{r}$$

Substituting the above into Eq (6.29),

$$\left(I_z+\frac{J_{eq}d}{2r^2}+mb^2\right)\dot{\omega}=-\frac{d^2}{2r^2}\left(B_{eq}+\frac{{n_g}^2K_tK_b}{R_a}\right)\omega-mbu\omega+\left(\frac{n_gK_td}{2R_ar}\right)\left(E_r-E_l\right)$$

$$\frac{R_a}{n_gK_t}\left(\frac{2rI_z}{d}+\frac{J_{eq}d}{r}+\frac{2mrb^2}{d}\right)\dot{\omega}-\frac{d}{r}\left(\frac{R_aB_{eq}}{n_gK_t}+n_gK_b\right)\omega+\frac{2R_amrb}{n_gK_td}u\omega=E_r-E_l \qquad (6.30)$$

6.2 Model Reference Adaptive Control

Adaptive control is a broad range of techniques that employ automatic tuning to help controllers sustain the performance levels of systems with parameters that vary over time or

are not fully known (Spong et al. 2006). The way that adaptive controllers work is based on the fundamental concept of online parameter estimation in which estimates of unknown or uncertain parameters are constantly fed back to the system to be used in the computation of the control input. The ability to cope with complex systems with parameter uncertainties makes this type of control a natural choice in the field of robotics. Adaptive control is inherently non-linear.

There are many types of adaptive control schemes, but they largely follow either of two approaches: indirect or direct (Landau 2003). In the first approach, also called the explicit method, the plant parameters are estimated online from measured data and then used to calculate the controller parameters. It strives towards the convergence of estimated and uncertain parameter values. In the second approach, also called the implicit method, controller parameters are estimated directly without any parameter estimation of the plant model. This means that there is no requirement for any explicit system identification. Thus, the direct approach leads to more straightforward designs as well as better performance in general. Nevertheless, each approach has its own advantages and disadvantages.

The control scheme adopted in this research is model reference adaptive control (MRAC). While MRAC can be of either the indirect of direct type, it is usually associated with the latter, and this will be the approach used in the research.

In MRAC systems, a reference model that describe the plant characteristics is used in the design of the controller (Slotine and Li 1991; Ioannou and Sun 1996). Basically, a control input is passed into both the reference model and plant, and the difference between the ideal and actual outcomes are used to make adjustments in the next set of inputs. A generalised MRAC system is shown in Fig. 6.1.

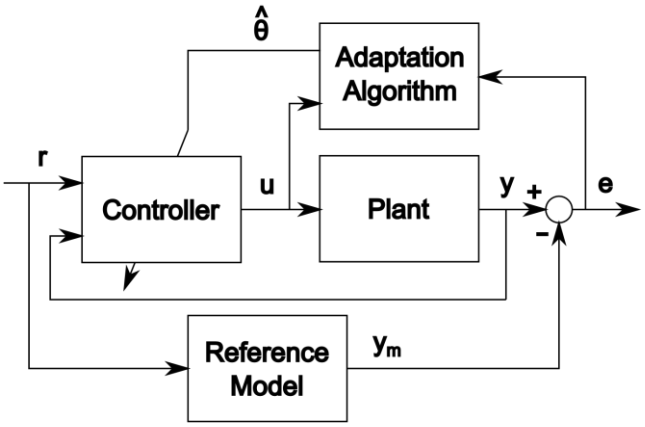


Fig. 6.1 Model Reference Adaptive Control System

In order to design the reference model, it is necessary to have an intimate understanding of the plant as well the level of performance required. Naturally, this involves detailed mathematical modelling. To ensure the stability of the system, Lyapunov's second (direct) method is used in the analysis.

The basic idea of Lyapunov's direct method is that if the total energy of a system is being dissipated continuously, it will eventually arrive at an equilibrium point and remain there (Freeman and Kokotović 1996).

The general procedure requires a suitable scalar energy-like function (also called the Lyapunov function) to be chosen and its derivative along system trajectories to be evaluated. If the derivative is demonstrated to be decreasing along the system trajectories as time is increasing, it can be concluded that the system's energy is dissipating and will thus settle eventually at equilibrium (Khalil 2002). If asymptotic stability can be proven using Lyapunov's method, there is no longer any necessity to solve for the system's differential equations to determine its stability. Hence, this approach is often called the direct method.

6.2.1 Feedback Linearisation

The rationale behind feedback linearisation is to transform complicated non-linear systems into simpler but equivalent linear forms. This is accomplished by a transformation of state and input, and with non-linear feedback. Linear systems are well-understood, extensively documented and relatively easier to design. So, there are clear advantages of being able to linearise a system.

In the model used here, the feedback control is organised in two parts as illustrated in Fig. 6.2. The first computes all the non-linearities within the plant so that the second can be designed to be based on a linear and decoupled system.

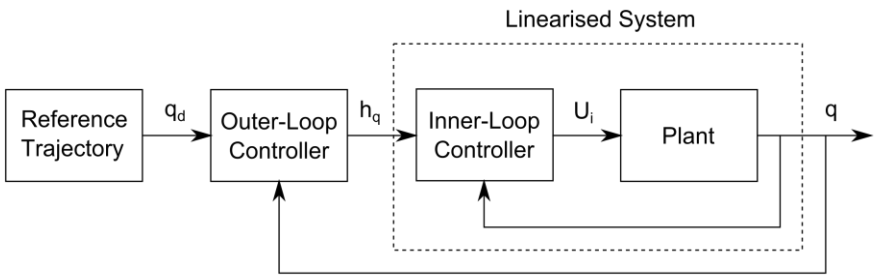


Fig. 6.2 Feedback Linearisation

A few common symbols that will be used from here on shall be defined now:

1. The subscript "d" is the desired value of an associated variable
2. The tilde (~) accent is the difference between current/nominal and desired values
3. The circumflex (^) accent is the nominal (estimated) value of a variable

The aim of this WMR is to track a trajectory, so it would seem intuitive to choose the coordinates [x, y] as the tracking vector. However, the control is more direct if it matches the physical behaviour of the non-holonomic WMR. As the WMR cannot move laterally to a prescribed position but rather has to steer towards it, a more direct tracking vector would include variables for longitudinal and turning (yaw) motion.

The kinematics of the system as described in Eqs (3.11) and (3.13) can be expressed in a matrix form:

$$\begin{bmatrix} u \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\dfrac{\sin\varphi}{a} & \dfrac{\cos\varphi}{a} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \tag{6.31}$$

Now, let the tracking vector be defined as:

$$\dot{q} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} u \\ \omega \end{bmatrix} \tag{6.32}$$

Following the above,

$$\ddot{q} = \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} \dot{u} \\ \dot{\omega} \end{bmatrix} \tag{6.33}$$

$$q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} x' \\ \varphi \end{bmatrix} \tag{6.34}$$

The dynamic equations in Eqs (6.28) and (6.30) can now be expressed in the form:

$$M\ddot{q} + C(\dot{q})\dot{q} = DU_i \tag{6.35}$$

where M is the inertial matrix, C is the matrix containing the Coriolis and centrifugal terms, D is a coefficient matrix, and U_i is the input vector, i.e.

$$M = \begin{bmatrix} \frac{R_a}{n_g K_t}\left(mr + \frac{2J_{eq}}{r}\right) & 0 \\ 0 & \frac{R_a}{n_g K_t}\left(\frac{2rI_z}{d} + \frac{J_{eq}d}{r} + \frac{2mrb^2}{d}\right) \end{bmatrix}$$

$$C(\dot{q}) = \begin{bmatrix} \frac{2}{r}\left(\frac{R_a B_{eq}}{n_g K_t} + n_g K_b\right) & -\frac{R_a mrb}{n_g K_t}\dot{q}_2 \\ \frac{2R_a mrb}{n_g K_t d}\dot{q}_2 & \frac{d}{r}\left(\frac{R_a B_{eq}}{n_g K_t} + n_g K_b\right) \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$$U_i = \begin{bmatrix} E_l \\ E_r \end{bmatrix} \qquad (6.36)$$

Choosing the control law with h_q as a new input that is yet to be defined,

$$DU_i = Mh_q + C(\dot{q})\dot{q}$$

$$\therefore \quad U_i = D^{-1}\left[Mh_q + C(\dot{q})\dot{q}\right] \qquad (6.37)$$

Substituting Eq (6.41) into Eq (6.39),

$$M\ddot{q} + C(\dot{q})\dot{q} = DD^{-1}\left[Mh_q + C(\dot{q})\dot{q}\right]$$

$$\therefore \quad \ddot{q} = h_q \qquad (6.38)$$

By choosing the input shown in Eq (6.37), the result is a system that is linear and decoupled as illustrated in Eq (6.38). This shows that the input h_q can now be carefully selected to control a SISO (single-input single-output) linear system.

Since there is a second order system involved, let the input h_q be:

$$h_q = \ddot{q}_d + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q} \qquad (6.39)$$

where $\tilde{q} = q_d - q$

$$\dot{\tilde{q}} = \dot{q}_d - \dot{q}$$

and λ is a strictly positive number.

$$\ddot{q}_d - h_q + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q} = 0$$
$$\ddot{q}_d - \ddot{q} + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q} = 0$$
$$\ddot{\tilde{q}} + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q} = 0 \qquad (6.40)$$

The above is the typical response of a critically-damped linear second order system with a natural frequency of λ. It shows that the error dynamics have two real poles and is thus exponentially convergent. The natural frequency is responsible for the speed of response of the system and also the rate of decay of the tracking error.

To solve for the acceleration of the robot,

$$\ddot{q} = M^{-1}\left[DU_i - C(\dot{q})\dot{q}\right] \qquad (6.41)$$

Substituting from Eq (6.38),

$$h_q = M^{-1}\left[DU_i - C(\dot{q})\dot{q}\right] \qquad (6.42)$$

Solving Eq (6.46),

$$Mh_q = DU_i - C(\dot{q})\dot{q}$$
$$U_i = D^{-1}\left[Mh_q + C(\dot{q})\dot{q}\right] \qquad (6.43)$$

The result is the same as Eq (6.37) and illustrates how the initial input was transformed within the system.

6.2.2 Linear Parameterisation and Adaptive Dynamics

In order to apply the equations of motion of any dynamic model for a purpose, the system parameters have to be identified and determined. These parameters could include masses, spatial dimensions, moments of inertia, etc. To help simplify analysis and solution, attempts are often made to linearise the equations of motion.

For example, the inertial, Coriolis and centrifugal terms of Eq (6.35) may be expressed in the following manner:

$$M\ddot{q} + C(\dot{q})\dot{q} = K_r(\dot{q}, \ddot{q})\theta_p \tag{6.44}$$

where K_r is the regressor matrix
θ_p is the parameter vector

It is clear that the original expression has been transformed, and is now linear in the parameters.

In inverse-dynamics, the values of system parameters are often uncertain or not known at all. This may be due to various kinds of errors or simply a lack of information. Thus, nominal (estimated) values of parameters are used instead.
Revising the control law from Eq (6.37),

$$U_i = D^{-1}\left[\hat{M}h_q + \hat{C}(\dot{q})\dot{q}\right] \tag{6.45}$$

where $\tilde{M} = M - \hat{M}$
$\tilde{C} = C - \hat{C}$

However, according to Eq (6.39),

$$h_q = \ddot{q}_d + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q}$$

Substituting Eq (6.39) into (6.45) and the result into Eq (6.35),

$$\begin{aligned}
M\ddot{q} + C(\dot{q})\dot{q} &= DD^{-1}\left[\hat{M}\left(\ddot{q}_d + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q}\right) + \hat{C}(\dot{q})\dot{q}\right] \\
\left(\tilde{M} + \hat{M}\right)\ddot{q} + C(\dot{q})\dot{q} &= \hat{M}\left(\ddot{q}_d + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q}\right) + \hat{C}(\dot{q})\dot{q} \\
\tilde{M}\ddot{q} + C(\dot{q})\dot{q} - \hat{C}(\dot{q})\dot{q} &= \hat{M}\left(\ddot{q}_d - \ddot{q} + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q}\right) \\
\tilde{M}\ddot{q} + \tilde{C}(\dot{q})\dot{q} &= \hat{M}\left(\ddot{\tilde{q}} + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q}\right) \\
\ddot{\tilde{q}} + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q} &= \hat{M}^{-1}\left[\tilde{M}\ddot{q} + \tilde{C}(\dot{q})\dot{q}\right]
\end{aligned} \tag{6.46}$$

Applying the technique of linear parameterisation as explained earlier, the expression consisting of the inertial, Coriolis and centrifugal terms in Eq (6.46) becomes:

$$\tilde{M}\ddot{q} + \tilde{C}(\dot{q})\dot{q} = K_r(\dot{q}, \ddot{q})\tilde{\theta}_p \tag{6.47}$$

where

$$K_r(\dot{q}, \ddot{q}) = \begin{bmatrix} \ddot{q}_1 & 0 & 2\dot{q}_1 & -\dot{q}_2{}^2 \\ 0 & \ddot{q}_2 & d\dot{q}_2 & \frac{2}{d}\dot{q}_1\dot{q}_2 \end{bmatrix} \tag{6.48}$$

and

$$\tilde{\theta}_p = \begin{bmatrix} \tilde{p}_1 \\ \tilde{p}_2 \\ \tilde{p}_3 \\ \tilde{p}_4 \end{bmatrix} = \begin{bmatrix} \frac{R_a}{n_g K_t}\left(mr + \frac{2J_{eq}}{r}\right) \\ \frac{R_a}{n_g K_t}\left(\frac{2rI_z}{d} + \frac{J_{eq}d}{r} + \frac{2mrb^2}{d}\right) \\ \frac{1}{r}\left(\frac{R_a B_{eq}}{n_g K_t} + n_g K_b\right) \\ \frac{R_a mrb}{n_g K_t} \end{bmatrix} \tag{6.49}$$

Thus, Eq (6.46) can be expressed as such:

$$\dddot{\tilde{q}} + 2\lambda\ddot{\tilde{q}} + \lambda^2\dot{\tilde{q}} = \hat{M}^{-1}\left[K_r(\dot{q}, \ddot{q})\tilde{\theta}_p\right] \tag{6.50}$$

Now, let the tracking error be defined as:

$$e_t = \begin{bmatrix} \tilde{q} \\ \dot{\tilde{q}} \end{bmatrix} = \begin{bmatrix} q_d - q \\ \dot{q}_d - \dot{q} \end{bmatrix} \tag{6.51}$$

The system equations can now be expressed in the standard state-space representation:

$$\dot{e}_t = Ae_t + B\left[\hat{M}^{-1}\left(K_r\tilde{\theta}_p\right)\right] \tag{6.52}$$

where

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\lambda_1{}^2 & 0 & -2\lambda_1 & 0 \\ 0 & -\lambda_2{}^2 & 0 & -2\lambda_2 \end{bmatrix} \tag{6.53}$$

and $$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.54}$$

Consider the Lyapunov function candidate:

$$V\left(e_t, \tilde{\theta}_p\right) = e_t{}^T P e + \tilde{\theta}_p{}^T \Gamma^{-1} \tilde{\theta}_p \tag{6.55}$$

where P and Γ are symmetric positive definite constant matrices and

$$PA + A^T P = -Q \tag{6.56}$$

where Q is strictly positive and symmetric, i.e.

$$Q = Q^T > 0 \tag{6.57}$$

Taking the derivative of Eq (6.55),

$$\dot{V} = -e_t Q e_t + 2\tilde{\theta}_p{}^T \left(\Gamma^{-1} \dot{\tilde{\theta}}_p + K_r{}^T \hat{M}^{-1} B^T P e_t \right) \tag{6.58}$$

For guaranteed stability, the last term is equated to zero. Hence,

$$\dot{\tilde{\theta}}_p = -\Gamma K_r{}^T \hat{M}^{-1} B^T P e_t \tag{6.59}$$

By definition:

$$\tilde{\theta}_p = \theta_p - \hat{\theta}_p \tag{6.60}$$

But since θ_p is constant,

$$\dot{\tilde{\theta}}_p = -\dot{\hat{\theta}}_p \tag{6.61}$$

Thus, the parameter update (adaptation) law is selected as such:

$$\dot{\hat{\theta}}_p = \Gamma K_r{}^T \hat{M}^{-1} B^T P e_t \tag{6.62}$$

Applying the above parameter update law to Eq (6.58),

$$\dot{V} = -e_t Q e_t \leq 0 \tag{6.63}$$

Since Q is strictly positive, it can be concluded that $\dot{V}\left(e_t, \tilde{\theta}_p\right)$ is negative semi-definite in the $\left(e_t, \tilde{\theta}_p\right)$ space, which means that V(t) ≤ V(0) for all t ≥ 0. Therefore, according to Barbalat's lemma, the position tracking errors, e_t, will converge to zero asymptotically and parameter errors will remain bounded. The detailed proof for the application of Barbalat's lemma has been described in existing literature (Popov 1973; Slotine and Li 1991; Fontes and Magni 2004) and is not shown here.

6.3 Kinematics and Geometry of Vehicular Motion

The navigational and tracking ability of a mobile robot to determine its location is not only dependent on the precision of its sensors. Equally important are the underlying algorithms that turn raw sensor data into useful and accurate information.

In the PID model, an Eulerian discretisation approach was used. When the sampling period is very short, the approximated results are near exact. It is the same method that is used when a robot has only one sensor and has to depend on kinematics to calculate its movement. In the adaptive model, a truer approach was adopted by utilising a geometrically accurate analysis.

Before an analytical framework can be formulated, it is imperative to gain a detailed understanding of the nature of how the sensors work. Although the type of optical mouse sensor used in this research has incredibly high precision, it also has its quirks and limitations.

When the sensor detects movement, it reports two sets of motion-related data: Δx and Δy. These figures correspond to the local displacement with respect to the sensor itself. Note that in this research, the default coordinates of the sensors are swapped to match those of the vehicle.

When the vehicle is turning, the sensor would move through a continuous series of arcs. Consider a segment that the sensor has covered over a random sampling period. Being that the path is an arc, a turning radius and centre must exist as shown in Fig. 6.3. The forward motion of any object in a circular motion is the tangent to the circle. The longitudinal (local x') axis of the sensor aligns with its forward motion, and therefore with the tangent as well. This means that its lateral (local y') axis coincides with the radius of the turning arc. Since the radius remains constant during the particular time period, it means that its displacement in the lateral (local y') direction is effectively zero. This implies that a sensor will not be able to distinguish between motion in a straight line and a perfect arc (or circle).

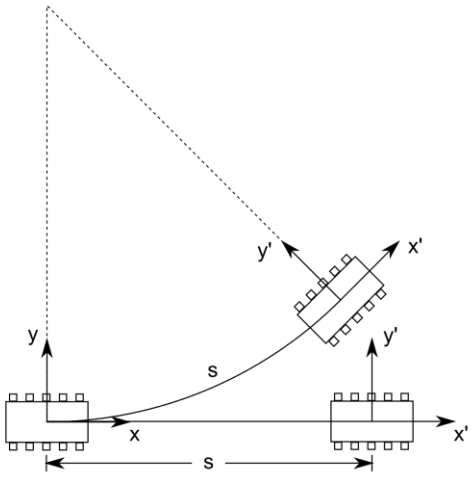


Fig. 6.3 Two distinct paths with identical sensor readings

A simple test will illustrate this particular behaviour of the sensor. The cord of a computer mouse is held down at any chosen position. That point on the cord being pressed down would represent a turning centre and the cord length between that point and the mouse would serve as a radius. This arrangement allows the mouse to move along the path of an arc. Instead of the pointer/cursor on the computer screen travelling along a similarly circular path, it will in fact move in an approximately straight line.

Based on the aforementioned characteristic, the same principle applies if there is an angular offset. As illustrated in Fig. 6.4, even if the longitudinal and lateral axes are not aligned with the tangent and radius respectively, the sensor can still be insensitive to turning motion. In this case, the constant offset angle contributes to a constant lateral (local y') displacement. This can be construed as constant lateral deviation and be treated as a constant gradient to a straight line. Therefore, any turning motion could still be potentially missed.

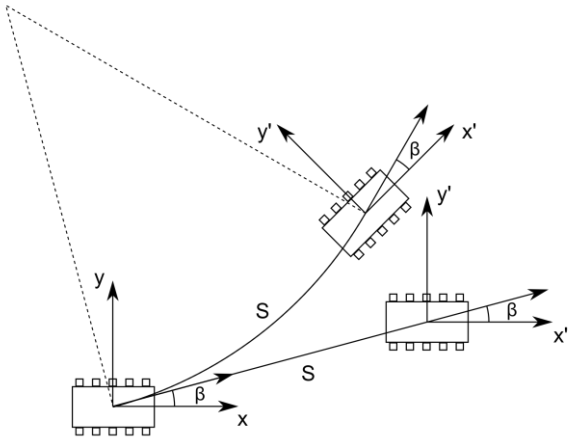


Fig. 6.4 Another example of two distinct paths with identical sensor readings

The use of two optical mouse sensors effectively negates the quirk of the sensor not being able to distinguish a straight path from a curved one. The different positions of the sensors mean that the each sensor will experience a different turning radius, and the corresponding tangents will not be equal. This is illustrated in Fig. 6.5. Using this piece of knowledge, the overall angular displacement of the vehicle can be worked out. Following that, the longitudinal and lateral displacements of each sensor can be derived geometrically. So, there is no longer any ambiguity in discerning a straight line from a curved path based on a two-sensor system. If both sensors have the same readings and angle between them is zero, the only conclusion is that the vehicle is travelling is a straight line in the longitudinal (local x') direction. There is absolutely no mistaking it for a curved trajectory.

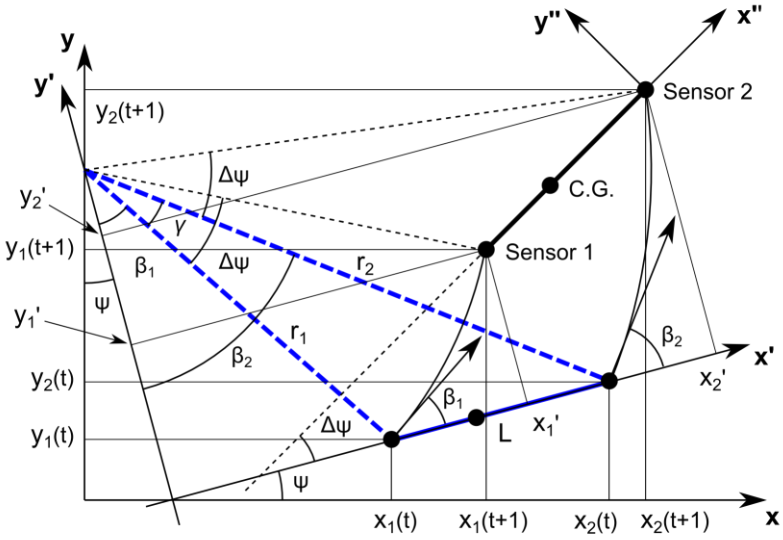


Fig. 6.5 Geometry of sensor movement during vehicle motion

In order to work out the geometry of the vehicle's path of travel, the known quantities have to be indentified first. As the sensors main purpose is to chart their own movement, it is only natural that distance travelled by each sensor is obtainable from their data output. The other known parameters are the vehicle's dimensions and how they relate to the sensors.

The geometrical calculation of the vehicle's path is straightforward if it is moving in a straight line. In any turning motion, there will be a common turning centre for the sensors and vehicle as a whole, but a different radius for each sensor because of how they are positioned. Since the path and direction travelled by each sensor can be derived from the output data, the magnitude of the turning radius can also be calculated - as will be shown in the equations below. As the distance between the sensors is a fixed constant, the triangle of three known sides can be formed as illustrated in Fig. 6.5. The obvious approach would be to use the Cosine Rule. This is similar to the method adopted by an earlier publication (Bonarini et al. 2004) for a WMR with a different sensor configuration. However, there are some key differences in the approach adopted in this thesis.

It has been established earlier that a two-sensor configuration removes any doubt in the straightness or curvature of a vehicle's trajectory. Although the shape of the vehicle's trajectory is no longer in question, the inherent nature of the individual sensor remains the same. This means that no matter whether the trajectory is a straight line or an arc, the Δx and Δy readings of each sensor will indicate the same displacement magnitudes. Refer to Figs. 6.3 and 6.4.

Therefore, the length of the path reported by each sensor can be calculated as a straight line:

$$s = \sqrt{(\Delta x')^2 + (\Delta y')^2} \tag{6.64}$$

The angle between local x' axis and tangent to the turning arc is:

$$\beta = \begin{cases} \tan^{-1}\left(\dfrac{\Delta y'}{\Delta x'}\right) & , \quad \Delta x' \neq 0 \\ \pm\dfrac{\pi}{2} & , \quad \Delta x' = 0 \end{cases} \tag{6.65}$$

The local longitudinal and lateral displacements are respectively:

$$\Delta x' = s\cos\beta \tag{6.66}$$

$$\Delta y' = s\sin\beta \tag{6.67}$$

The fixed distance between sensors is:

$$L = k - a \tag{6.68}$$

where a is the distance between Sensor 1 and the wheel baseline
k is the distance between Sensor 2 and the wheel baseline

Using the Cosine Rule,

$$L^2 = r_1^{\,2} + r_2^{\,2} - 2r_1 r_2 \cos\gamma \tag{6.69}$$

where the angle between the turning radii is:

$$\gamma = \left|\beta_2 - \beta_1\right| \tag{6.70}$$

Knowing the arc length, the radius of the turning arc can be determined:

$$s_1 = r_1\left|\Delta\psi\right| \tag{6.71}$$

$$r_1 = \frac{s_1}{\left|\Delta\psi\right|} \tag{6.72}$$

$$s_2 = r_2\left|\Delta\psi\right| \tag{6.73}$$

$$r_2 = \frac{s_2}{\left|\Delta\psi\right|} \tag{6.74}$$

By substituting Eqs (6.72) and (6.74) into the Cosine Rule of Eq (6.69), the change in heading (yaw) can be found:

$$\Delta\psi = \begin{cases} \dfrac{\sqrt{s_1^{\,2} + s_2^{\,2} - 2s_1 s_2 \cos\gamma}}{L} & , \quad \Delta y_1 < \Delta y_2 \\ -\dfrac{\sqrt{s_1^{\,2} + s_2^{\,2} - 2s_1 s_2 \cos\gamma}}{L} & , \quad \Delta y_1 > \Delta y_2 \\ 0 & , \quad \Delta y_1 = \Delta y_2 \end{cases} \tag{6.75}$$

For computational purposes,

$$r_{1,signed} = \frac{s_1}{\Delta\psi} \tag{6.76}$$

$$r_{2,signed} = \frac{s_2}{\Delta\psi} \tag{6.77}$$

It is clear that in Eqs (6.72), (6.74), (6.76) and (6.77), if the yaw angle Δψ is zero, the turning radii r_1 and r_2 will become infinitely long. From a physical point of view, it indicates that the vehicle is moving in a straight line purely along its longitudinal x-axis. Since the vehicle is not turning in any way, there is no turning radius. Thus, the magnitudes of the two turning radii present in the geometric equations will become mathematically infinite. Hence, care must be taken to formulate the equations in such a way as to accommodate this scenario.

As angle Δψ tends towards zero and the turning radii grow infinitely large, there will come a point when computational limits are reached in terms of variable size and processing power. Thus, a decision must be made on the precision level governing when to round off Δψ as zero as it becomes infinitesimally small. Special-case algorithms would then be invoked to handle the scenario of Δψ being zero. Naturally, small quantisation and rounding errors will occur as a result.

$$\Delta x_1' = \begin{cases} r_{1,signed}\left[\cos\left(\frac{\pi}{2} - \beta_1 - \Delta\psi\right) - \cos\left(\frac{\pi}{2} - \beta_1\right)\right] & , \quad \Delta\psi \neq 0 \\ \Delta x_A & , \quad \Delta\psi = 0 \end{cases}$$

$$\Rightarrow \quad \Delta x_1' = \begin{cases} r_{1,signed}\left[\sin(\beta_1 + \Delta\psi) - \sin\beta_1\right] & , \quad \Delta\psi \neq 0 \\ \Delta x_A & , \quad \Delta\psi = 0 \end{cases} \tag{6.78}$$

$$\Delta y_1' = \begin{cases} -r_{1,signed}\left[\sin\left(\frac{\pi}{2} - \beta_1 - \Delta\psi\right) - \sin\left(\frac{\pi}{2} - \beta_1\right)\right] & , \quad \Delta\psi \neq 0 \\ 0 & , \quad \Delta\psi = 0 \end{cases}$$

$$\Rightarrow \quad \Delta y_1' = \begin{cases} r_{1,signed}\left[\cos\beta_1 - \cos(\beta_1 + \Delta\psi)\right] & , \quad \Delta\psi \neq 0 \\ 0 & , \quad \Delta\psi = 0 \end{cases} \tag{6.79}$$

$$\Delta x_2' = \begin{cases} r_{2,signed}\left[\cos\left(\frac{\pi}{2} - \beta_2 - \Delta\psi\right) - \cos\left(\frac{\pi}{2} - \beta_2\right)\right] & , \quad \Delta\psi \neq 0 \\ \Delta x_K & , \quad \Delta\psi = 0 \end{cases}$$

$$\Rightarrow \quad \Delta x_2' = \begin{cases} r_{2,signed}\left[\sin(\beta_2 + \Delta\psi) - \sin\beta_2\right] & , \quad \Delta\psi \neq 0 \\ \Delta x_K & , \quad \Delta\psi = 0 \end{cases} \tag{6.80}$$

$$\Delta y_2' = \begin{cases} -r_{2,signed}\left[\sin\left(\frac{\pi}{2}-\beta_2-\Delta\psi\right)-\sin\left(\frac{\pi}{2}-\beta_2\right)\right] & , \quad \Delta\psi \neq 0 \\ 0 & , \quad \Delta\psi = 0 \end{cases}$$

$$\Rightarrow \qquad \Delta y_2' = \begin{cases} r_{2,signed}\left[\cos\beta_2 - \cos\left(\beta_2+\Delta\psi\right)\right] & , \quad \Delta\psi \neq 0 \\ 0 & , \quad \Delta\psi = 0 \end{cases} \tag{6.81}$$

Note that Δx_A, Δy_A, Δx_K and Δy_K are the reported displacement data along the local axis as reported by the Sensors 1 and 2 at positions A and K respectively. These numbers are not equivalent to actual local displacements $\Delta x_1'$, $\Delta y_1'$, $\Delta x_2'$ and $\Delta y_2'$ respectively, except when there is no turning motion, i.e. $\Delta\psi$ is zero. As discussed previously, since the sensors are insensitive to turning motion, the perceived displacement values may not reflect the actual path taken. The true local displacements, Δx_1, Δy_1, Δx_2 and Δy_2, have to be determined by geometric calculations.

Using the similarity principle, the local displacement of the centre of gravity can be found:

$$\Delta x_G' = \frac{b-a}{k-a}\left\{\left[\Delta x_2'+(a-b)\right]-\left[\Delta x_1'+(k-b)\right]\right\}+\left[\Delta x_1'+(k-b)\right] \tag{6.82}$$

$$\Delta y_G' = \frac{b-a}{k-a}\left\{\left[\Delta y_2'+(a-b)\right]-\left[\Delta y_1'+(k-b)\right]\right\}+\left[\Delta y_1'+(k-b)\right] \tag{6.83}$$

The local velocities at the WMR's centre of mass are simply derived from Eqs (6.82) and (6.83):

$$u = \frac{dx_G'}{dt} \tag{6.84}$$

$$v = \frac{dy_G'}{dt} \tag{6.85}$$

With a heading angle of ψ known from the previous sampling period, the global displacements of the sensors can be calculated as thus:

$$\Delta x_{1,\,global} = \begin{cases} r_{1,signed}\left[\sin\left(\beta_1+\Delta\psi+\psi\right)-\sin\left(\beta_1+\psi\right)\right] & , \quad \Delta\psi \neq 0 \\ \Delta x_A\cos\psi & , \quad \Delta\psi = 0 \end{cases} \tag{6.86}$$

$$\Delta y_{1,\,global} = \begin{cases} r_{1,signed}\left[\cos\left(\beta_1+\psi\right)-\cos\left(\beta_1+\Delta\psi+\psi\right)\right] & , \quad \Delta\psi = 0 \\ 0 & , \quad \Delta\psi = 0 \end{cases} \tag{6.87}$$

$$\Delta x_{2,\,global} = \begin{cases} r_{2,signed}\left[\sin\left(\beta_2 + \Delta\psi + \psi\right) - \sin\left(\beta_2 + \psi\right)\right] & , \quad \Delta\psi \neq 0 \\ \Delta x_K \cos\psi & , \quad \Delta\psi = 0 \end{cases} \tag{6.88}$$

$$\Delta y_{2,\,global} = \begin{cases} r_{2,signed}\left[\cos\left(\beta_2 + \psi\right) - \cos\left(\beta_2 + \Delta\psi + \psi\right)\right] & , \quad \Delta\psi = 0 \\ 0 & , \quad \Delta\psi = 0 \end{cases} \tag{6.89}$$

Since Sensor 2 is the designated reference sensor for the WMR's position,

$$x\left(t+1\right) = x\left(t\right) + \Delta x_{2,\,global} \tag{6.90}$$

$$y\left(t+1\right) = y\left(t\right) + \Delta y_{2,\,global} \tag{6.91}$$

$$\psi_{new} = \psi + \Delta\psi \tag{6.92}$$

6.4 Limitations of the Geometric Approach

The analysis of the WMR's motion presented in the previous section is the ideal geometric model. It is certainly a more faithful mathematical representation of a vehicle's motion than the linear approximation approach of Euler as discussed in Chapter 3. However, the ideal method is not without its limitations - especially when applied computationally. As pointed out in the previous section, there are inevitable truncation and rounding errors as well as constraints on variable size and processing capabilities.

In the later chapter describing the prototype, the documentation will show that the main program uses double-precision (64-bit) floating point numbers for storage and calculations. This is necessary because the high precision of the sensor leads to very small values during calculations, and the turning radius' tendency to approach infinity means that numbers can also get very large. Indeed, simulations confirm that the use of single-precision computation was inadequate for the program to execute without a significant loss of precision in handling either very small or very large numbers.

A major shortcoming of the main microcontroller used by the WMR is that it has no floating point unit. This is quite typical of microcontrollers used in embedded systems. As such, every non-integer number has to be handled via emulation. This inevitably leads to a significant loss of processor performance. Depending on the operation, it could be slower than similar integer calculations by several orders of magnitude.

Aside from the processor limitations, there is another notable concern. The direct consequence of very small numbers being calculated is the large magnitude of the turning radius. Small fluctuations in the raw sensor data, whether caused by detection error or sensor

noise, may cause a similarly small error percentage in the calculated value of the turning radius. However, the large magnitudes involved mean that even a tiny percentage shift can lead to a significant deviation in the final calculated values in absolute terms. In essence, errors are magnified. This was clearly noticed during prototype testing.

During the prototype testing phase, the calibration of sensors revealed that there was a slight difference between them. On a typical test surface, the sensor with the lower sensitivity managed 3.045 x 10^{-5} m/count. The minimum detectable vehicle turning angle is when either (but not both) of the sensors registers a single count in the lateral (Δy') direction. Using either Eq (3.18) or Eq (6.79), the minimum turning resolution of the vehicle can be determined.

$$\Delta\psi_{\min} = \pm 3.383\times10^{-4}\ rad \tag{6.93}$$

So, it is theoretically impossible for the program to return a value smaller than the above. However, this is not always the case according to experimental data. Regardless, as Δψ approaches zero and the program needs to round off to zero in order to invoke the special-case algorithm which presumes the vehicle to be travelling in an absolutely straight trajectory. A cut-off point is required, and the minimum turning resolution serves as a good criterion for it. By including a generous margin of error, the cut-off point was set at:

$$\Delta\psi_{\min} = 4.0\times10^{-4}\ rad \tag{6.94}$$

Once this minimum point was set, the turning radii of both sensors will no longer continue to approach infinity. Yet, data collected from all the test runs reveal that both radii routinely exceed 2 x 10^{13} m and occasionally even surpass 3 x 10^{13} m. These are indeed very large numbers when compared to the size of the vehicle and typical travelling distance.

It is known fact that real numbers are not represented exactly in a binary system that is typically used in modern computing (Goldberg 1991). Their approximate representations are called floating point numbers, and there is a tendency for a loss of precision as numbers get very large. This is especially true if those large numbers are derived from very small values which could lead to a loss of significance.

The processors and software used in the development of this prototype all adhere to the IEEE Standard for Floating-Point Arithmetic (IEEE 754). The standard measure of precision is the unit in the last place (ULP). This is the smallest possible difference between two consecutive numbers that can be represented. It is akin to a resolution or granularity metric, and is relative to the magnitude of a number. The smallest possible precision is measured at a value of "1" and is called the machine epsilon.

The determination of the ULP at a given numerical range is quite straightforward. MATLAB also has a built-in function to calculate the ULP which is called eps(). Since the native computing format is in binary, the calculations will use a base of two. For a given number, n,

$$ULP(n) = \beta^{e-(p-1)} \tag{6.95}$$

where β is the base
e is the integer exponent
p is the number of precision bits

In a double-precision (64-bit) number, there are 53 (52 stored) significand bits, 11 exponent bits, and 1 sign bit. The range of numbers from 2 x 10^{13} to 3 x 10^{13} can be represented in base 2 as $2^{44.19}$ to $2^{44.77}$. For any number somewhere within this said range, the ULP is:

$$ULP(2\times10^{13}) = 2^{44-(53-1)}$$
$$ULP(2\times10^{13}) = 3.906\times10^{-3} \tag{6.96}$$

As mentioned before, 2 x 10^{13} m is a value frequently exceeded in calculations of the radius length. This radius is directly used to determine the coordinates of the sensors and by extension the vehicle. While the sensors have precision levels at about 3 x 10^{-5} m/count, the large radii values used in the calculations can only manage a precision of just under 4 x 10^{-3} m. This loss of precision amounts to more than two orders of magnitude (i.e. over 100 times) and is thus considered very significant. This is certainly a serendipitous discovery. The problem is compounded by the fact that it happens quite frequently when the vehicle is travelling in nearly a straight line. Furthermore, dead-reckoning errors are cumulative.

Indeed, if the sensor resolution were set higher than the nominal 800 cpi, the minimum detectable angle would be smaller, thus allowing the maximum turning radius to reach a far larger magnitude before it has to be truncated. For example, if the maximum radius were to be increased by a factor of 10, the precision of the calculated coordinates would be in the range of well over 15 mm.

$$ULP(1\times10^{14}) = 1.563\times10^{-2}$$

As is clearly illustrated here, bigger numbers lead to a greater loss of precision. Ironically, in order to achieve the highest precision possible, accuracy is actually lost in the process.

Thus, it is clear that although the geometric model is mathematically accurate, it is not quite so in a practical sense. Furthermore, there are other issues in addition to the limitations in

floating-point precision. To avoid situations leading to infinity, very small numbers have to be rounded to zero. At the other end of the scale, extremely large values have to be truncated. So, the ideal model is not as perfect as expected. Nevertheless, the geometric approach is still a more accurate method than Euler's linear approximation in general.

The rounding and quantisation errors as well as noise sensitivity issues of the geometric method only occur within a small range of motion, i.e. when the turning angle $\Delta\psi$ is near or equal to zero. Other than that, the aforementioned concerns do not apply. However, vehicles do generally move forward in somewhat of a straight line much of the time. So, although the problem is confined to a narrow range of motion, it occurs quite frequently. In contrast, Euler's method suffers from discretisation errors throughout the entire range of motion except when the vehicle is travelling in an absolutely straight line.

6.5 Euler's Method Revisited and a Hybrid Technique

It is clear that as the geometric method approaches a point where its precision begins to deteriorate, it is also when Euler's method tends to be most accurate. The logical conclusion would be to combine the two methods. The geometric approach would cover nearly the entire range of motion, while Euler's method would apply to situations where turning is nearly or totally absent. The key question would be when to switch between methods.

An obvious choice for the switch-over point is just before the turning radius reaches a magnitude where the numerical precision of the processor falls below the sensor's resolution. For rounding purposes, the critical number should be no more than half the resolution of the sensor with the higher sensitivity.

Let r_{crit} be the critical radius at which computing precision exceeds sensor resolution, σ_s. Thus, the minimum allowable computing precision is:

$$ULP(r_{crit}) < 0.5\sigma_s \tag{6.97}$$

But $\beta = 2$ for any binary system. Hence,

$$\begin{aligned}
&\beta^{e_{crit}-(p-1)} < 0.5\sigma_s \\
&2^{e_{crit}-(p-1)} < 0.5\sigma_s \\
&e_{crit} - (p-1) < \log_2(0.5\sigma_s) \\
&e_{crit} < \log_2(0.5\sigma_s) + p - 1
\end{aligned} \tag{6.98}$$

where e_{crit} is the base-2 exponent that corresponds to the critical radius

However, the ULP is calculated using rounded-down integer exponents. This means that any number up to but not including the next higher integer is rounded down to the next lower integer. So, the exclusive limit of e_{crit} is the next higher integer. A common function in C and MATLAB used for performing this rounding-up operation is called ceil(). Thus, Eq (6.102) should be revised as such:

$$e_{crit} < ceil\left[\log_2(0.5\sigma_s) + p - 1\right] \tag{6.99}$$

The critical radius can now be determined:

$$r_{crit} < 2^{e_{crit}} \tag{6.100}$$

According to the calibration results in Table 8.3, the more sensitive sensor of the two has a resolution of 2.87 x 10^{-5} m. To calculate the critical radius of the system,

$$e_{crit} < ceil\left[\log_2(0.5 \times 2.87 \times 10^{-5}) + 53 - 1\right]$$
$$e_{crit} < 36$$
$$r_{crit} < 2^{36}$$
$$r_{crit} < 6.872 \times 10^{10}$$

The critical radius will thus be set at:

$$r_{crit} = 6.8 \times 10^{10}\ m \tag{6.101}$$

The corresponding computer (machine) precision at this numerical range is confirmed to be well beneath the sensor's resolution:

$$ULP(6.8 \times 10^{10}) = 7.629 \times 10^{-6} \tag{6.102}$$

Based on the above figures, the switch-over from the geometric technique to Euler method was assigned to be the point when either of the radii exceeds 6.8 x 10^{10} m. The tracking equations can now be revised accordingly. For convenience, Eqs (6.64) to (6.77) will be reproduced here albeit without their original descriptions and explanations:

$$s = \sqrt{(\Delta x')^2 + (\Delta y')^2} \tag{6.103}$$

$$\beta = \begin{cases} \tan^{-1}\left(\dfrac{\Delta y'}{\Delta x'}\right) & , \quad \Delta x' \neq 0 \\ \pm\dfrac{\pi}{2} & , \quad \Delta x' = 0 \end{cases} \tag{6.104}$$

$$\Delta x' = s\cos\beta \tag{6.105}$$

$$\Delta y' = s\sin\beta \tag{6.106}$$

$$L = k - a \tag{6.107}$$

$$L^2 = r_1^{\,2} + r_2^{\,2} - 2r_1 r_2\cos\gamma \tag{6.108}$$

$$\gamma = \left|\beta_2 - \beta_1\right| \tag{6.109}$$

$$s_1 = r_1\left|\Delta\psi\right| \tag{6.110}$$

$$r_1 = \frac{s_1}{\left|\Delta\psi\right|} \tag{6.111}$$

$$s_2 = r_2\left|\Delta\psi\right| \tag{6.112}$$

$$r_2 = \frac{s_2}{\left|\Delta\psi\right|} \tag{6.113}$$

$$\Delta\psi = \begin{cases} \dfrac{\sqrt{s_1^{\,2} + s_2^{\,2} - 2s_1 s_2\cos\gamma}}{L} & , \quad \Delta y_1 < \Delta y_2 \\ -\dfrac{\sqrt{s_1^{\,2} + s_2^{\,2} - 2s_1 s_2\cos\gamma}}{L} & , \quad \Delta y_1 > \Delta y_2 \\ 0 & , \quad \Delta y_1 = \Delta y_2 \end{cases} \tag{6.114}$$

$$r_{1,signed} = \frac{s_1}{\Delta\psi} \tag{6.115}$$

$$r_{2,signed} = \frac{s_2}{\Delta\psi} \tag{6.116}$$

Referring back to Fig. 6.5, the revised equations are thus:

$$\Delta x_1' = \begin{cases} r_{1,signed}\left[\sin(\beta_1+\Delta\psi)-\sin\beta_1\right] & , \quad r_1 \le r_{crit} \\ s_1\cos\beta_1 & , \quad r_1 > r_{crit} \end{cases}$$

Or,

$$\Delta x_1' = \begin{cases} r_{1,signed}\left[\sin(\beta_1+\Delta\psi)-\sin\beta_1\right] & , \quad r_1 \le r_{crit} \\ \Delta x_A & , \quad r_1 > r_{crit} \end{cases} \tag{6.117}$$

$$\Delta y_1' = \begin{cases} r_{1,signed}\left[\cos\beta_1-\cos(\beta_1+\Delta\psi)\right] & , \quad r_1 \le r_{crit} \\ s_1\sin\beta_1 & , \quad r_1 > r_{crit} \end{cases}$$

Or,

$$\Delta y_1' = \begin{cases} r_{1,signed}\left[\cos\beta_1-\cos(\beta_1+\Delta\psi)\right] & , \quad r_1 \le r_{crit} \\ \Delta y_A & , \quad r_1 > r_{crit} \end{cases} \tag{6.118}$$

$$\Delta x_1' = \begin{cases} r_{2,signed}\left[\sin(\beta_2+\Delta\psi)-\sin\beta_2\right] & , \quad r_2 \le r_{crit} \\ s_2\cos\beta_2 & , \quad r_2 > r_{crit} \end{cases}$$

Or,

$$\Delta x_2' = \begin{cases} r_{2,signed}\left[\sin(\beta_2+\Delta\psi)-\sin\beta_2\right] & , \quad r_2 \le r_{crit} \\ \Delta x_K & , \quad r_2 > r_{crit} \end{cases} \tag{6.119}$$

$$\Delta y_1' = \begin{cases} r_{2,signed}\left[\cos\beta_2-\cos(\beta_2+\Delta\psi)\right] & , \quad r_2 \le r_{crit} \\ s_2\sin\beta_2 & , \quad r_2 > r_{crit} \end{cases}$$

Or,

$$\Delta y_2' = \begin{cases} r_{2,signed}\left[\cos\beta_2-\cos(\beta_2+\Delta\psi)\right] & , \quad r_2 \le r_{crit} \\ \Delta y_K & , \quad r_2 > r_{crit} \end{cases} \tag{6.120}$$

The equations for the local displacement of the centre of gravity remain the same as Eqs (6.82) and (6.83):

$$\Delta x_G' = \frac{b-a}{k-a}\left\{\left[\Delta x_2'+(a-b)\right]-\left[\Delta x_1'+(k-b)\right]\right\}+\left[\Delta x_1'+(k-b)\right] \tag{6.121}$$

$$\Delta y_G' = \frac{b-a}{k-a}\left\{\left[\Delta y_2'+(a-b)\right]-\left[\Delta y_1'+(k-b)\right]\right\}+\left[\Delta y_1'+(k-b)\right] \tag{6.122}$$

Similarly, the local velocities at the WMR's centre of mass are unchanged from Eqs (6.84) and (6.85):

$$u = \frac{dx_G'}{dt} \tag{6.123}$$

$$v = \frac{dy_G'}{dt} \tag{6.124}$$

The revised global displacements of the sensors are:

$$\Delta x_{1,\,global} = \begin{cases} r_{1,signed}\left[\sin\left(\beta_1 + \Delta\psi + \psi\right) - \sin\left(\beta_1 + \psi\right)\right] & , \quad r_1 \le r_{crit} \\ s_1 \cos\left(\beta_1 + \psi\right) & , \quad r_1 > r_{crit} \end{cases} \tag{6.125}$$

Or, $$\Delta x_{1,\,global} = \begin{cases} r_{1,signed}\left[\sin\left(\beta_1 + \Delta\psi + \psi\right) - \sin\left(\beta_1 + \psi\right)\right] & , \quad r_1 \le r_{crit} \\ \Delta x_A \cos\psi - \Delta y_A \sin\psi & , \quad r_1 > r_{crit} \end{cases} \tag{6.126}$$

$$\Delta y_{1,\,global} = \begin{cases} r_{1,signed}\left[\cos\left(\beta_1 + \psi\right) - \cos\left(\beta_1 + \Delta\psi + \psi\right)\right] & , \quad r_1 \le r_{crit} \\ s_1 \sin\left(\beta_1 + \psi\right) & , \quad r_1 > r_{crit} \end{cases} \tag{6.127}$$

Or, $$\Delta y_{1,\,global} = \begin{cases} r_{1,signed}\left[\cos\left(\beta_1 + \psi\right) - \cos\left(\beta_1 + \Delta\psi + \psi\right)\right] & , \quad r_1 \le r_{crit} \\ \Delta x_A \sin\psi + \Delta y_A \cos\psi & , \quad r_1 > r_{crit} \end{cases} \tag{6.128}$$

$$\Delta x_{2,\,global} = \begin{cases} r_{2,signed}\left[\sin\left(\beta_2 + \Delta\psi + \psi\right) - \sin\left(\beta_2 + \psi\right)\right] & , \quad r_2 \le r_{crit} \\ s_2 \cos\left(\beta_2 + \psi\right) & , \quad r_2 > r_{crit} \end{cases} \tag{6.129}$$

Or, $$\Delta x_{2,\,global} = \begin{cases} r_{2,signed}\left[\sin\left(\beta_2 + \Delta\psi + \psi\right) - \sin\left(\beta_2 + \psi\right)\right] & , \quad r_2 \le r_{crit} \\ \Delta x_K \cos\psi - \Delta y_K \sin\psi & , \quad r_2 > r_{crit} \end{cases} \tag{6.130}$$

$$\Delta y_{2,\,global} = \begin{cases} r_{2,signed}\left[\cos\left(\beta_2 + \psi\right) - \cos\left(\beta_2 + \Delta\psi + \psi\right)\right] & , \quad r_2 \le r_{crit} \\ s_2 \sin\left(\beta_2 + \psi\right) & , \quad r_2 > r_{crit} \end{cases} \tag{6.131}$$

Or, $$\Delta y_{2,\,global} = \begin{cases} r_{2,signed}\left[\cos\left(\beta_2 + \psi\right) - \cos\left(\beta_2 + \Delta\psi + \psi\right)\right] & , \quad r_2 \le r_{crit} \\ \Delta x_K \sin\psi + \Delta y_K \cos\psi & , \quad r_2 > r_{crit} \end{cases} \tag{6.132}$$

The equations for the WMR's position are still the same as Eqs (6.90) to (6.92). Assigning Sensor 2 as the reference sensor,

$$x\left(t+1\right) = x\left(t\right) + \Delta x_{2,\,global} \tag{6.133}$$

$$y\left(t+1\right) = y\left(t\right) + \Delta y_{2,\,global} \tag{6.134}$$

$$\psi_{new} = \psi + \Delta\psi \tag{6.135}$$

The above equations do not include any accommodation for sensor misalignment. These misalignment angles are obtained through calibration and can simply be included later. The next section provides an example of how this is done. With minor modifications, the general principle of the method described above is applicable to other systems that employ multiple optical mouse sensors.

6.6 Error Detection and Correction for Partial Redundancy

The accuracy of the navigation and tracking of a WMR depends on the precision of its sensors as well as the localisation algorithm used. The previous section underlined the practical limitations of both method and computation precision, and how they can be accurately addressed. However, without reliable sensor data, all other efforts will amount to nothing. This section will discuss the identification of sensor errors and the means to cope with them while maintaining an acceptable level of tracking accuracy.

It is commonly understood that even under the best experimental circumstances, sensors will occasionally report inaccurate data. Presume that either of the sensors has provided erroneous readings which can be identified and corrected (or discarded). If the reliable data from the other sensor could be used as some sort of reference for correction, then some level of sensor redundancy could be achieved. It is impossible for data from a single sensor to fully compensate for the loss of data from the other sensor. Factors such as slippage cannot be accounted for. Thus, the redundancy level can only be considered partial.

Many WMRs past and present have only relied on the use of one sensor for navigation via dead-reckoning. These types of systems generally depend on a kinematic approach for coordinate calculation. The inherent quirks of the optical mouse sensor as well as the inability to fully account for slippage make it unsuitable for precise localisation. While the kinematic approach may not possess the best level of accuracy, it is still widely used in robotics for situations where slippage minimal. So, it is postulated that the kinematic method could still be used sparingly as a contingency algorithm when data from one of the sensors is faulty. It is certainly preferable to using questionable data or stopping the vehicle completely until its position can be reset.

There are several causes leading to data variance and error in a sensor, such as surface texture, clearance between sensor and surface, sensor velocity, and random noise (Minoni and Signorini 2006; Palacin et al. 2006). Sensor noise here is a collective term for unpredictable errors such electrical fluctuations as well as other uncertainties. It is important to identify and account for this type of error, but it cannot be eliminated.

In the research, only one type of surface is used per series of tests. The surface is carefully selected and cleaned to ensure that it is quite uniform and visibly free defects or debris. While not perfect, the average characteristics of the surface can be considered to be rather consistent. Thus, any significant change in sensor sensitivity due to this reason would be infrequent, localised and limited to an extremely short span of time.

In regards to the effect of sensor speed, there is a small variability as long as the sensor operates within the manufacturer's recommendation. Since the WMR used in this research is only capable of very low acceleration and the sampling time is quite small (i.e. 0.02 s), the vehicle speed can be seen as almost constant in between polling. Thus, the effect of sensor speed is presumed to be marginal. Several initial straight-line tests at different speeds (30% to 100% of maximum) detected no discernible difference in measured distance.

There is one factor that contributes by far the biggest variability in sensor sensitivity and thus has the most influence over the data values. It is the distance between the sensor and the surface that it is tracking. This is known from the hardware charts provided by the manufacturer as well as from the experience gained during the calibration process.

Although the recommended operating height range is very narrow, newer sensors have a wider margin outside that range where they still perform near optimum level. Beyond the extended range, sensitivity changes drastically and generally drops off precipitously. That is why it is extremely important to ensure that the test surface is relatively flat and void of defects.

In order to maintain a constant ground clearance, some researchers have affixed the mouse (in its original enclosure) to the WMR such that it is in constant contact with the ground. Naturally, this arrangement causes a lot of friction and may obstruct the vehicle's movement if there is even the slightest surface protrusion. If the mouse manages to scale the obstruction, then it has lifted off the surface and has thus altered the distance between sensor and ground. The same could happen if any of the wheels were to roll over a small object or debris. This negates the benefit of having the mouse slide on the ground. The only clear advantage of this design is that minimum distance between the sensor and surface is constrained by the bottom thickness of the mouse, and thus there is a clear lower limit.

Generally, many of WMRs that use optical mouse sensors have a small ground clearance so that the sensor (or its enclosure) would not rub along the ground and thus allow for smoother running. This research uses the same set-up. However, the trade-off for this sort of configuration is that there is a greater fluctuation in the distance between sensor and surface, and thus a greater potential for inaccurate readings.

In the calculation of sensor data to determine the vehicle's position, the distance between the two sensors is a known and constant parameter. So, this serves as an excellent reference in any calculation for detecting data variance. Now, the two sensors on this WMR have a fixed distance between them on the same longitudinal axis, so their displacement readings in the longitudinal direction should ideally be identical. If there is any difference in their readings on the shared axis, their relative positions to each other on that axis will change. This means that

the distance between the sensors is no longer the same as before. Since this is a physically impossibility, the original assumption holds true. Hence, for Sensor 1 located at position A, and Sensor 2 located at position K, their local displacements are:

$$\Delta x_A = \Delta x_K \tag{6.136}$$

Including adjustments for minor misalignment,

$$\Delta x_A \cos \varepsilon_A = \Delta x_K \cos \varepsilon_K \tag{6.137}$$

where ε is the misalignment angle with respect to the longitudinal axis, x'

So, if Eq (6.137) does not hold true, one or both readings may be incorrect. Naturally, a certain margin of error must be allowed. If the readings in the x-axis are questionable, the readings in the y-axis from the same sample must also be treated similarly. It is perfectly possible for both readings to be erroneous and still pass the Eq (6.137) condition. However, this is coincidental and unlikely to happen often. If this situation does indeed arise, it would be quite impossible to detect. Furthermore, there would be no means to resolve the problem because this method requires at least one sensor to report reliable readings at any given time.

A previous study (Bonarini et al. 2005) assumed that the errors encountered in the authors' research were due to under-reporting that is caused by a decrease in sensitivity levels. This implies that displacement values should never exceed what are expected. In that research, two mice in their original shells were affixed to the WMR in such a way that they were forced to make contact with the ground. So, the height between the sensor and tracking surface can never be less than the bottom thickness of the mouse housing. According to the manufacturer's hardware charts (Fig. 8.14), the sensitivity levels decrease when the sensor height exceeds the optimum range. So, for that research, the hypothesis seems perfectly reasonable. Unfortunately, these assumptions do not apply to other WMRs that maintain a gap between sensor and ground.

In this research, the sensor's ground clearance can vary such that it can be more or less than the optimum height. According to the manufacturer's data (Fig. 8.15) of the mouse used in this research, sensitivity levels can be higher of lower than the optimum values outside the recommended height range. The principle behind the workings of this type of sensors suggests that once out of the optimum/working zone, sensitivity levels will generally decrease as height increases. In this case, as the sensitivity levels can vary both ways, it means that erroneous readings could be more or less than the actual displacements. Ultimately, the sensitivity levels are expected to decrease as height increases.

Now, even if data inconsistency can be detected using Eq (6.137), it still remains to be determined which sensor is responsible for the fault. Fortunately, in addition to displacement values, the sensor also reports the number of features tracked by the sensor in each frame. This metric is called "surface quality" or SQUAL for short. Generally, sensor tracking improves with a greater number of identifiable features. Conversely, if a surface is perfectly smooth and featureless, or if the distance between sensor and surface is beyond the sensor's focal range, there will be no identifiable features. In such cases, the sensor will report no displacement readings and the SQUAL value would be zero.

A low SQUAL value may not necessarily mean that the data is unreliable. The SQUAL values of a sensor would likely be relatively low if the tracking surface has a texture that is rather homogeneous. However, the sensor may still be able identify enough features to track properly. On the other hand, if the sensor is unable to track with any consistency, other than a hardware malfunction, it is almost certain that the SQUAL values would be very low or near zero.

With this knowledge in mind, the SQUAL data could be used in conjunction with Eq (6.137) in order to design a test for identifying erroneous data. Firstly, the SQUAL data for each sensor on a given test surface are recorded during sensor calibration. Like the sensitivity values, the SQUAL data is also averaged over the number of samples as shown in Table 8.4. The results show that the SQUAL readings are quite consistent over the same surface. This will serve as a benchmark for good quality readings.

The two sensors report slightly different SQUAL values over the same surface. So, it is meaningless to use a direct comparison between the two as a test for data reliability. Instead, each sensor's data should be weighed against its calibrated reference. In statistics, 99.74% of a sample set is contained within three standard deviations (σ). The common practice is to consider any data that falls outside this region as an outlier. So, this is the convention will be followed here.

Due to experimental error, it is unrealistic to use Eq (6.137) as a test without including a margin of error. Again, the data from the calibrations can be used to determine the average difference between the readings of both sensors as shown in Table 8.4. This will be used as a reference for judging data reliability. Similar to the SQUAL data, the error margin used here will be three standard deviations.

Restating the fundamental test,

$$\left|\Delta x_A \cos\varepsilon_A - \Delta x_K \cos\varepsilon_K\right| \le 3\sigma_x \tag{6.138}$$

If this condition is met within the designated allowance, the calculations will simply follow the procedure described in Section 6.5. However, the moment the condition fails, the immediate task would be to identify the sensor with the presumably faulty readings. The following equations are to be used until the next set of data is sampled and put through the test again.

The SQUAL test used to determine whether data is considered reliable is:

$$SQUAL(Sensor) \leq \mu_{SQUAL} + 3\sigma_{SQUAL} \tag{6.139}$$

where μ_{SQUAL} is the reference mean SQUAL value for the sensor
σ_{SQUAL} is the reference standard deviation for the sensor

Consider the scenario where the data from the front (second) sensor has failed the SQUAL test and is deemed unreliable. Accounting for any misalignment by including a correctional angle, ε, the local and global displacements can be determined in the equations below.

The local displacements of Sensor 1 are:

$$\Delta x_1' = \Delta x_A \cos\varepsilon_A - \Delta y_A \sin\varepsilon_A \tag{6.140}$$

$$\Delta y_1' = \Delta x_A \sin\varepsilon_A + \Delta y_A \cos\varepsilon_A \tag{6.141}$$

As explained before, the sensor makes no distinction between a straight-line or curved trajectory. Ignoring any potential slippage, the turning centre can only fall somewhere along the wheel baseline and its outward projections on both ends. The turning angle and velocity are thus:

$$a(\Delta\psi) = \Delta y_1'$$

$$\Rightarrow \quad \Delta\psi = \frac{\Delta y_1'}{a} \tag{6.142}$$

$$\omega = \frac{\Delta\psi}{\Delta t} \tag{6.143}$$

where Δt is the sampling period

The data from Sensor 2 has been discarded for this sample period, so the data from Sensor 1 must be extrapolated to estimate the coordinates at the location of Sensor 2. It has already been demonstrated that the displacement in the local x-axis are the same for both sensors.

Also, the turning angle is common for the entire vehicle. Using the principle of similarity, the displacement in the local y-axis can be projected.

$$\Delta x_2' = \Delta x_1' \tag{6.144}$$

$$\Delta y_2' = k(\Delta \psi) \tag{6.145}$$

To calculate dynamic forces acting on the WMR, the local displacements and velocities at its centre of gravity must be determined. The same method for projecting the data for Sensor 2 is applied here.

$$\Delta x_G' = \Delta x_1' \tag{6.146}$$

$$\Delta y_G' = b(\Delta \psi) \tag{6.147}$$

$$u = \frac{\Delta x_G'}{\Delta t} \tag{6.148}$$

$$v = \frac{\Delta y_G'}{\Delta t} \tag{6.149}$$

Since Sensor 2 is the designated reference point, the global displacements will be calculated with respect to this sensor's location.

$$\Delta x_2 = \Delta x_2 \cos\psi - \Delta y_2 \sin\psi \tag{6.150}$$

$$\Delta y_2 = \Delta x_2 \sin\psi + \Delta y_2 \cos\psi \tag{6.151}$$

Finally, the equations for the WMR's global position are:

$$x(t+1) = x(t) + \Delta x_{2,} \tag{6.152}$$

$$y(t+1) = y(t) + \Delta y_2 \tag{6.153}$$

$$\psi_{new} = \psi + \Delta\psi \tag{6.154}$$

Now, consider the alternative case where the rear (first) sensor's data has failed the SQUAL condition and is considered unacceptable. The calculations for the local displacements of Sensor 2 are analogous to the previous scenario.

$$\Delta x_2' = \Delta x_K \cos\varepsilon_K - \Delta y_K \sin\varepsilon_K \tag{6.155}$$

$$\Delta y_2' = \Delta x_K \sin\varepsilon_K + \Delta y_K \cos\varepsilon_K \tag{6.156}$$

Similarly, the turning angle and velocity can be determined like before:

$$k(\Delta\psi) = \Delta y_2'$$

$$\Rightarrow \quad \Delta\psi = \frac{\Delta y_2'}{k} \tag{6.157}$$

$$\omega = \frac{\Delta\psi}{\Delta t} \tag{6.158}$$

Since Sensor 2 is the designated reference point, there is no need to calculate the local displacements or velocities of Sensor 1. As before, the local displacements and velocities at the WMR's centre of gravity can be determined by using the principle of similarity.

$$\Delta x_G' = \Delta x_2' \tag{6.159}$$

$$\Delta y_G' = b(\Delta\psi) \tag{6.160}$$

$$u = \frac{\Delta x_G'}{\Delta t} \tag{6.161}$$

$$v = \frac{\Delta y_G'}{\Delta t} \tag{6.162}$$

The global displacements based on Sensor 2 as the designated reference point is the same as in the previous case.

$$\Delta x_2 = \Delta x_2 \cos\psi - \Delta y_2 \sin\psi \tag{6.163}$$

$$\Delta y_2 = \Delta x_2 \sin\psi + \Delta y_2 \cos\psi \tag{6.164}$$

Like before, the equations for the WMR's global position are:

$$x(t+1) = x(t) + \Delta x_{2,} \tag{6.165}$$

$$y(t+1) = y(t) + \Delta y_2 \tag{6.166}$$

$$\psi_{new} = \psi + \Delta\psi \tag{6.167}$$

The shortcomings of the kinematic approach have already been extensively documented. So, this algorithm should only used sparingly as a back-up option when data from one of the sensors is deemed unreliable. The moment the data from both sensors have passed the fitness test, the tracking algorithm should be switched back to the hybrid approach that was discussed in the preceding section.

6.7 Summary

A heavily-revamped mathematical model of the wheeled robot has been developed and presented in this chapter. While it has relied on some of the theoretical foundation established in prior chapters, the bulk of this model differs from the previous version to the extent that it could almost be considered a new design.

The current model is a dynamic system that employs model reference adaptive control in contrast to the PID controllers used previously. In theory, the self-tuning capabilities of the adaptive controller provide superior performance and versatility. Thus, it will be expected to address some of the problems encountered by the earlier model.

A crucial part of the wheel robot is the sensor system. Hence, it was necessary to highlight the peculiar traits of the optical mouse sensors that are planned for use on the prototype. Another important area to investigate is the different methods for calculating odometry. Both kinematic and geometric approaches were analysed, and their respective advantages and disadvantages were identified. By combining the best attributes of both odometric methods, a new hybrid technique was formulated.

Lastly, a novel partial-redundancy system was devised to compensate for momentary sensor error or loss of tracking. With this new model complete, it will now need to be converted into a computational representation so that it can be tested in a simulated environment.

Chapter 7: Computational Design of the Adaptive Model

The theoretical underpinnings of the adaptive model are far removed from that of the earlier PID system. Thus, the resulting model has almost been entirely reworked and resembles little of the previous system. Using the mathematical foundation established in the Chapters 3 and 6, a new computational representation will be developed in this chapter. Once again, the simulation is performed with the help of the MATLAB® program and its graphical modelling tool, Simulink®. When the simulation results are satisfactory, modifications would be made in order for it to be compiled for the processor of the WMR and work with its hardware components.

7.1 General Outline of the Computational Model

The basic idea of this design is similar to that of the previous PID model and other systems in general. A trajectory generator issues a number of checkpoints along a desired path for the WMR to follow. The sensors detect the displacements at every sampling time which are then used to determine the coordinates and heading at the present location. Knowing the distance between the current position and the next checkpoint as well as the time allowed for the WMR to travel to the next location, its speed and direction can be calculated. The full model of the system is shown in Fig 7.1. For greater clarity, an enlarged version is placed in Appendix A.

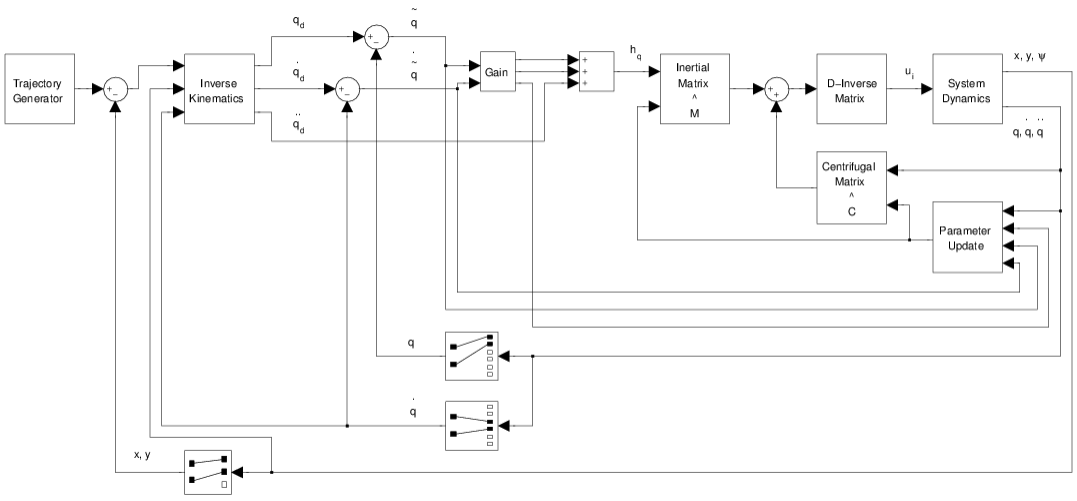


Fig. 7.1 Adaptive Tracking Control System of the WMR

7.2 Assumptions and Limitations of the Simulation Model

The general assumptions and limitations in the mathematical model in Chapters 3 and 6 also apply here. Raw sensor data cannot be fully mimicked, so the required translational and turning (yaw) speeds are fed back into a subsystem containing motor and vehicle parameters in order to calculate the projected displacements. Sensor noise is not included in the simulation. Also, since this is an ideal condition, there is no slippage involved. Thus, this is an ideal scenario that tests the controller's performance and not those of the sensors.

7.3 Design Details of the Simulation Model

The simulation process of the adaptive model is similar to that of the PID system. The algorithms that describe a theoretical model are converted into computational code and run over and over again to produce data. This output data is then compared with the predicted outcome to see how well they match up.

There are a few differences in the approach here compared to the previous model. Initially, the adaptive model was formulated as a continuous system according to the theoretical equations. After it was constructed and simulated tests had been run, it was then converted into a discrete model. As the actual WMR is a discrete system, this model is more realistic. When the discrete model had been fully tested in simulation, it was then modified such that the simulated data was replaced by an interface for the sensors.

7.3.1 Trajectory Generator

The purpose of a trajectory generator is to discretise a predetermined path into a series of points that coincide with every sampling instance. As the WMR is required to cover the distance between points at each sample, the element of time is thus introduced. As such, the created path becomes a trajectory. The Trajectory Generator subsystem as shown in Fig 7.2 is the only subsystem that is taken directly from the PID model almost unchanged. Further elaboration can be found in Chapter 4. Additional trajectories are included for the adaptive model. The scope, graph and matrix-file-output blocks are present in the simulation version of the model. However, in order to conserve memory, they are removed from the final model compiled for the WMR's microcontroller.

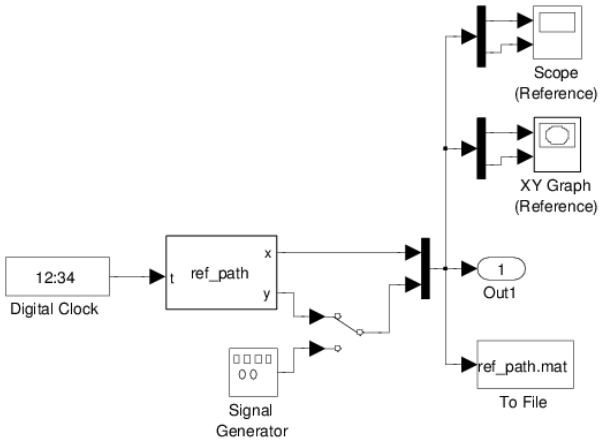
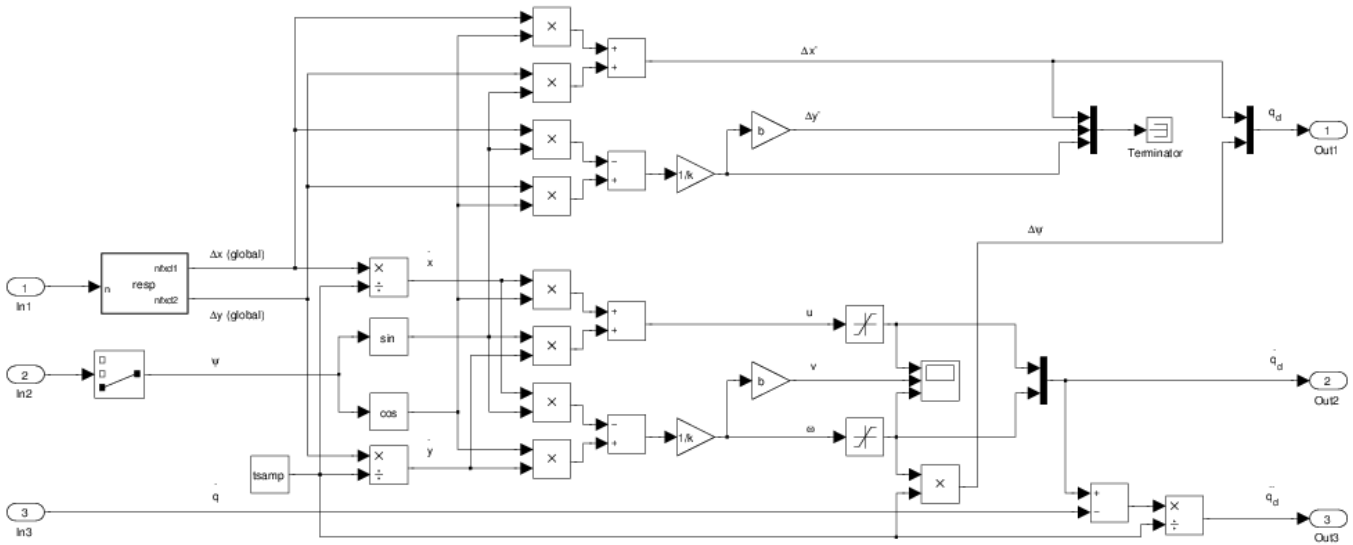


Fig. 7.2 Trajectory Generator Subsystem

7.3.2 Inverse Kinematics

In order for the controller to determine the required speed and direction to get to the next trajectory point, it requires information on the discrepancy between the current location of the WMR and the desired position at that particular time. This is the purpose of the Inverse Kinematics subsystem shown in Fig. 7.3. The theory behind the equations used in the calculations here are described in Chapter 3. In the PID system, the required displacements in the X- and Y-axes are used to calculate how fast each wheel would need to rotate. In the adaptive model, those displacement values are used to calculate the tracking variables, which are the longitudinal and turning velocities. The determination of wheel speed is not calculated in this subsystem but determined later in the model.



b: Distance between wheel baseline and centre of mass
a: Distance between wheel baseline and Sensor 1 at Point A
k: Distance between wheel baseline and Sensor 2 at Point K
Δx: Displacement in global x–coordinate
Δy: Displacement in global y–coordinate
$\dot{x}$: Velocity in global x–coordinate
$\dot{y}$: Velocity in global y–coordinate
Δx': Displacement in local x'–coordinate
Δy': Displacement in local y'–coordinate
u: Velocity in local x'–coordinate
v: Velocity in local y'–coordinate
ψ: Vehicle orientation in global coordinates
Δψ: Vehicle yaw
ω: Vehicle yaw rate
t_{samp}: Sampling period
q_d: Desired tracking vector

$\dot{q}_d$: Derivative of desired tracking vector

$\ddot{q}_d$: Double derivative of desired tracking vector

Fig. 7.3 Inverse Kinematics Subsystem

7.3.3 Gain Subsystem

The Gain subsystem consists mainly of a strictly positive number that is associated with the system's natural frequency λ. In various combinations, this value makes up the positive definite matrices for the positional and velocity gains. The schematic of the subsystem is illustrated in Fig. 7.4.

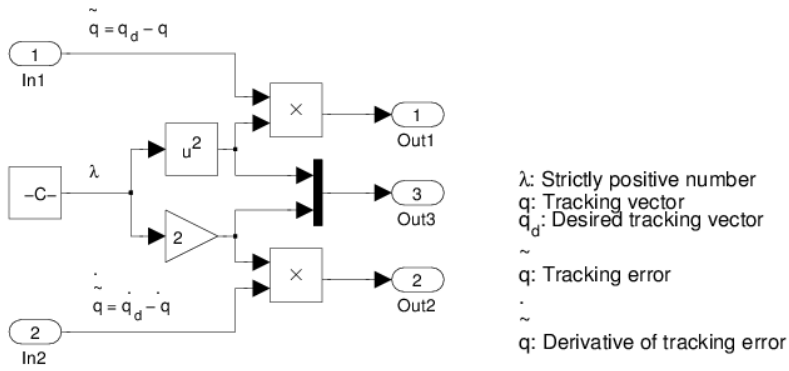


Fig. 7.4 Gain Subsystem

7.3.4 Inertial Matrix

The Inertial Matrix subsystem as shown in Fig. 7.5 consists of the inertial terms of the dynamic equations of the system as described by Eqs (6.35) and (6.36). The coefficients of the inertial terms are linearly parameterised according to Eqs (6.47) to (6.49) and are constantly updated via the adaptive mechanism.

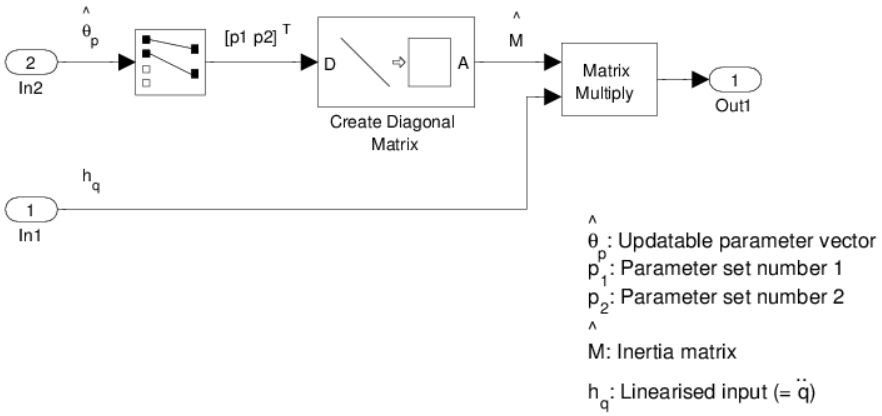


Fig. 7.5 Inertial Matrix Subsystem

7.3.5 Centrifugal Matrix

The Centrifugal Matrix subsystem as shown in Fig. 7.6 contains both the centrifugal and Coriolis terms of the system's dynamic equations as described by Eqs (6.35) and (6.36). Like the inertial matrix, the coefficients of the both terms are also linearly parameterised according to Eqs (6.47) to (6.49) and are updated continuously by the adaptive algorithm.

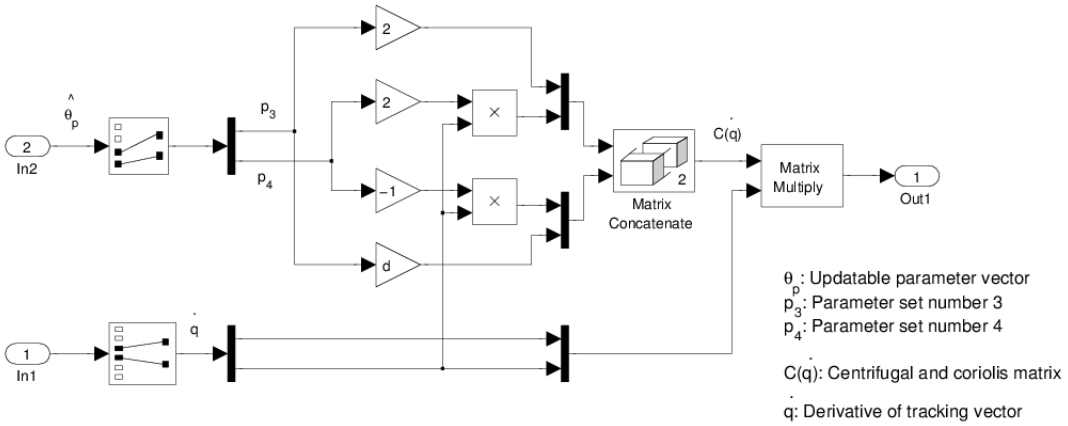


Fig. 7.6 Centrifugal Matrix Subsystem

7.3.6 D-Inverse Matrix

The D matrix as described by Eqs (6.35) and (6.36) consists of nothing more than the coefficients of the voltage inputs of the linearised system. The D- Inverse matrix (or D^{-1}) is just a mathematic manipulation of the equations for deriving the input vector as shown in Eq (6.37). The subsystem is illustrated in Fig. 7.7.

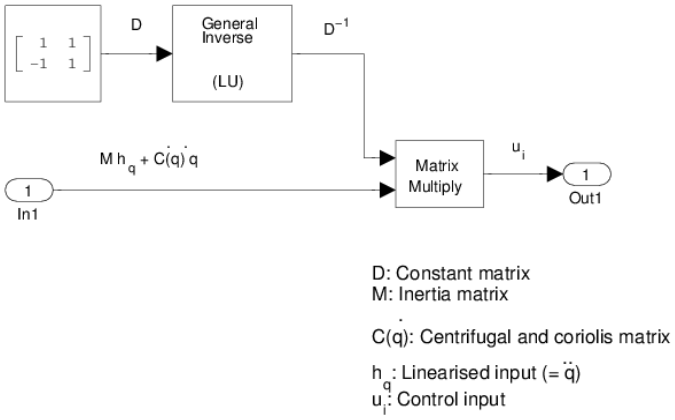


Fig. 7.7 D-Inverse Matrix Subsystem

7.3.7 Parameter Update

One of the defining characteristics of adaptive control is the updatable nature of its parameters. These parameters are tuned with the use of Lyapunov stability analysis in response to the discrepancy between the actual and desired outcomes of the system. As illustrated in Eqs (6.47) to (6.49), the dynamic equations of the system can be linearly parameterised and arranged in a vector form. The resulting parameters are dynamically tuned in the Parameter Update subsystem as shown in Fig. 7.8. The adaptive gain settings are configured within a symmetric positive definite matrix as represented by the Constant Diagonal Matrix block.

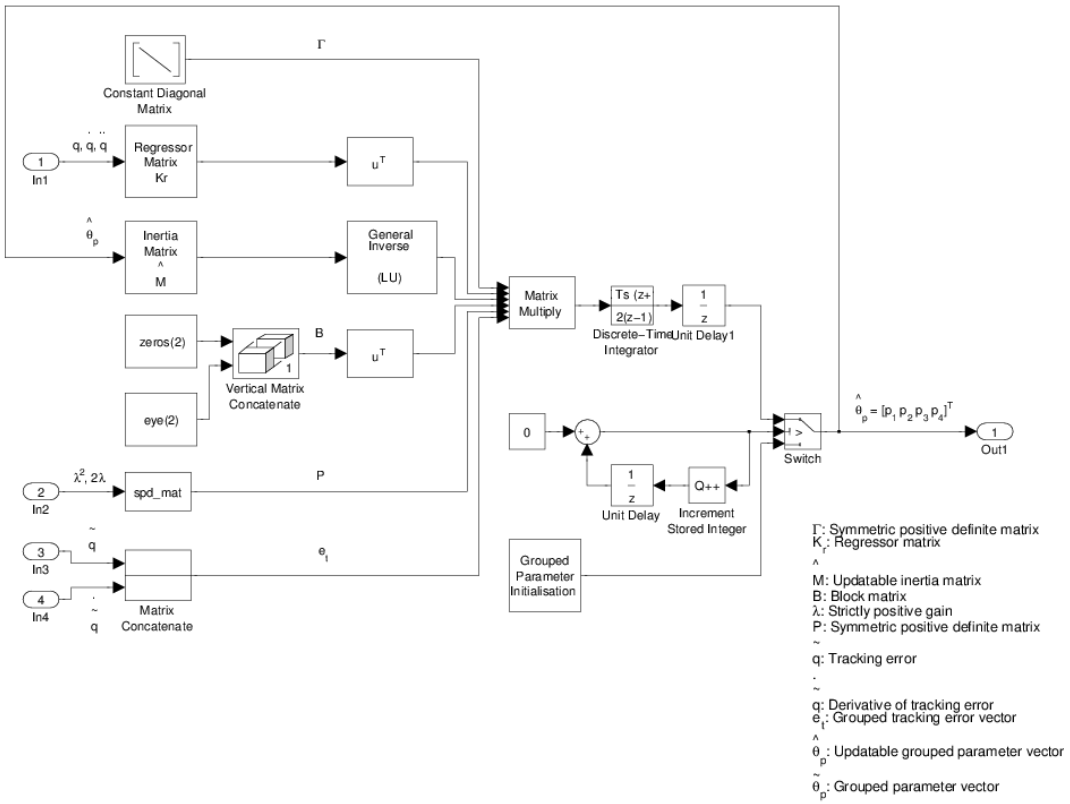


Fig. 7.8 Parameter Update Subsystem

7.3.7.1 Grouped Parameter Initialisation

At the start of the program, the updatable control parameters need to have initial values. Each parameter is itself made up of a combination of several other system characteristics. Some of these properties can be accurately measured or calculated, while others may be approximated. Together, they provide nominal or estimated figures that are used for parameter initialisation. These values do not have to be exact since the adaptive algorithm

will correct them as the program is running. However, good estimates help speed up the convergence process. All the initial parameter values are grouped together as a vector in a single subsystem as shown in Fig. 7.9.

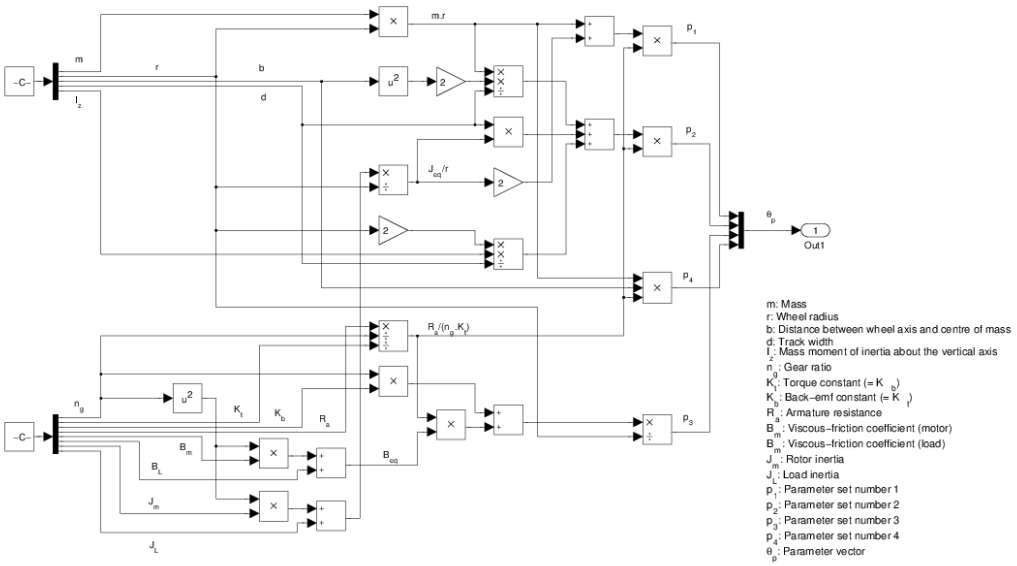


Fig. 7.9 Grouped Parameter Initialisation Subsystem

7.3.7.2 Regressor Matrix

It has been shown previously that the dynamic equations of the system can be linearly parameterised into a regressor function and parameter vector as demonstrated in Eqs (6.47) to (6.49). The Regressor Matrix subsystem is shown in Fig. 7.10.

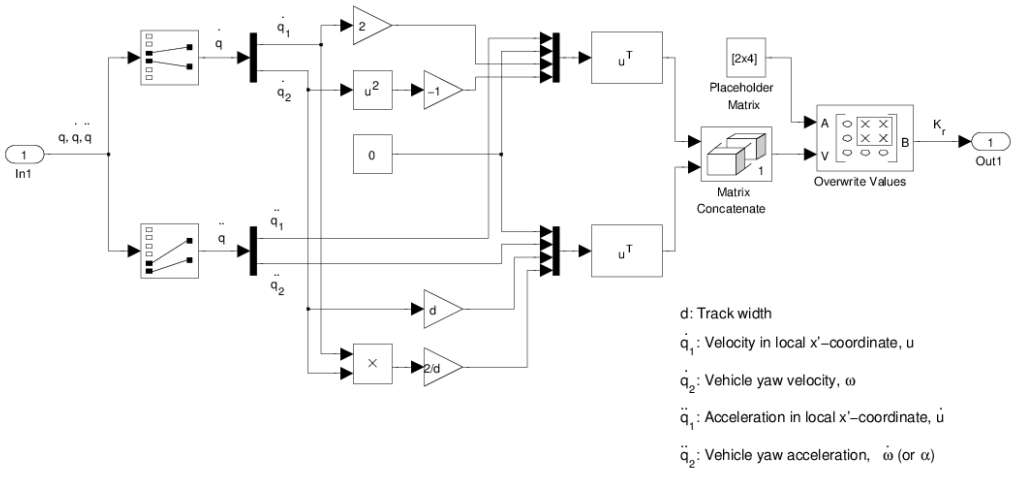


Fig. 7.10 Regressor Matrix Subsystem

7.3.7.3 Symmetric Positive Definite Matrix (spd_mat) Function

The calculation of a symmetric positive definite constant matrix, P, is required in the derivation of the Lyapunov function candidate as depicted in Eqs (6.55) to (6.63). This matrix is determined with the help of MATLAB's lyap() function which is embedded in an S-Function block. The relevant code is placed in Appendix A. Unfortunately, the function cannot be ported to the compiled program for the WMR's microcontroller. Therefore, the values generated by the simulation model are manually entered into hardware model before compilation. Despite the slight inconvenience, this approach works because the numbers in this matrix remain constant throughout the program's execution.

7.3.8 System Dynamics

The System Dynamics subsystem as shown in Fig 7.11 represents the non-linear model of the wheeled robot. In the simulation version, voltage inputs are used in the theoretical derivation of the linear and yaw accelerations of the WMR. In turn, these accelerations are used in the simulation of the displacements that are expected from a hypothetical sensor. With the displacements known, the final coordinates can be determined. The Grouped Parameters, Inertia Matrix and Centrifugal Matrix subsystems are identical to the similarly-named ones described in preceding sections.

In the version compiled for the WMR's microcontroller, all the aforementioned subsystems along with the simulated sensor are replaced with subsystems that interface with actual hardware components like the DC motors and optical mouse sensors.

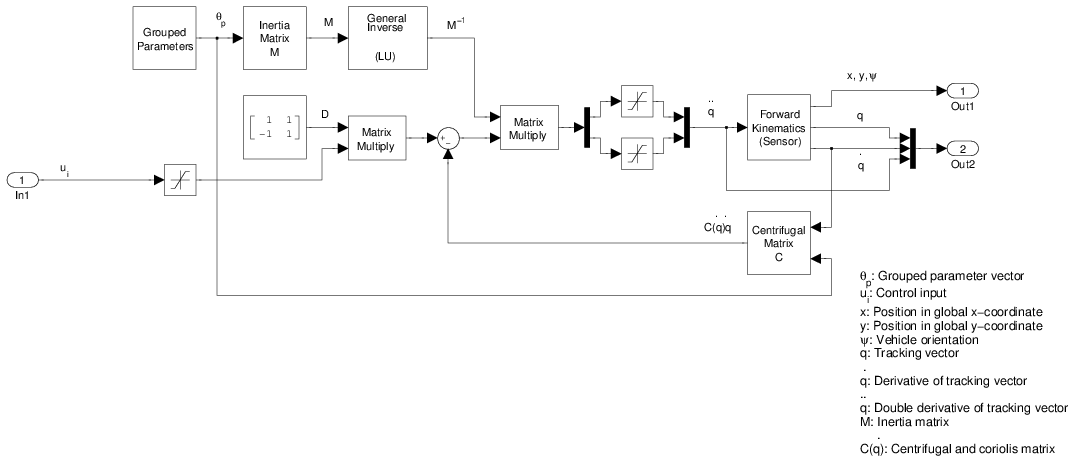
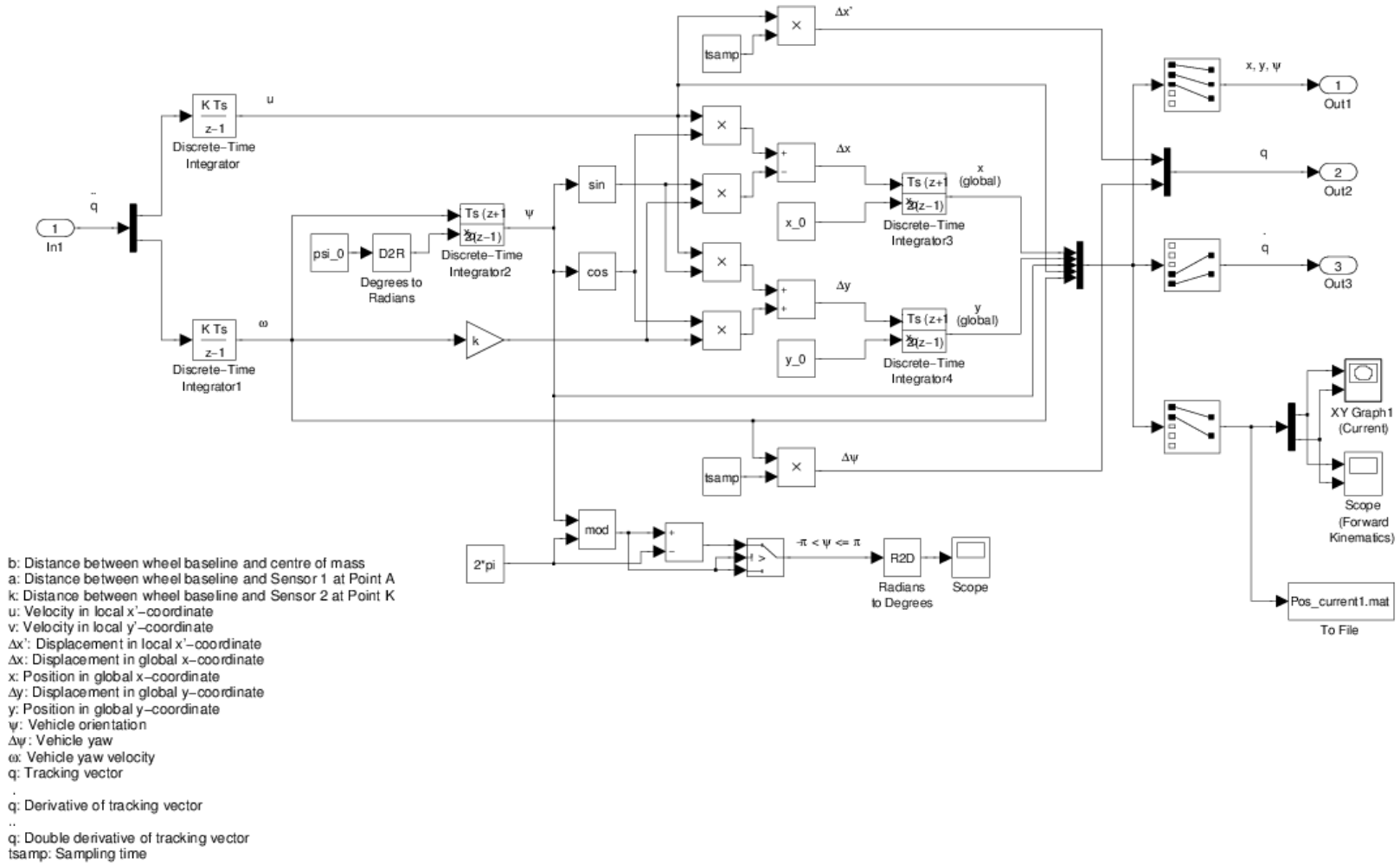


Fig. 7.11 System Dynamics Subsystem

7.3.8.1 Forward Kinematics (Sensor)

Just as in the PID system, the Forward Kinematics subsystem in the adaptive model represents a simulated analogue for the optical mouse sensor. As it is impossible to simulate the displacement data from an optical sensor, forward kinematics is used to calculate the odometry of the WMR. This simulation technique relies on dead reckoning, which is the same method used by the WMR to process actual sensor data. Since the sampling period is very small and slippage is not a factor during simulation, this approach is deemed valid.

In this subsystem as shown in Fig. 7.12, localisation is accomplished by applying forward kinematics to the lateral and yaw accelerations generated by the simulated system dynamics. The results are then used to determine the displacement of the WMR with respect to the global coordinates during the sampling period.



b: Distance between wheel baseline and centre of mass
a: Distance between wheel baseline and Sensor 1 at Point A
k: Distance between wheel baseline and Sensor 2 at Point K
u: Velocity in local x'−coordinate
v: Velocity in local y'−coordinate
Δx': Displacement in local x'−coordinate
Δx: Displacement in global x−coordinate
x: Position in global x−coordinate
Δy: Displacement in global y−coordinate
y: Position in global y−coordinate
ψ: Vehicle orientation
Δψ: Vehicle yaw
ω: Vehicle yaw velocity
q: Tracking vector
.
q: Derivative of tracking vector
..
q: Double derivative of tracking vector
tsamp: Sampling time

Fig. 7.12 Forward Kinematics (Sensor) Subsystem

7.4 Design Details of Deployed Model

The deployed model as shown in Fig. 7.13 is the version of the program that actually interfaces with hardware components such as the DC motors and optical mouse sensors. Its design is mostly similar to the simulated version with one key exception. As the program is actually interfacing with real hardware peripherals, it no longer needs to simulate sensor data or the WMR's movement. Hence, the System Dynamics subsystem, which is used for simulation calculations, is no longer needed. In its place is the System Interface subsystem that contains blocks which communicate with the hardware components. Additionally, a slight change had to be made to the Parameter Update subsystem. Other than that, all other subsystems from the simulation model are carried over to the deployed version unchanged.

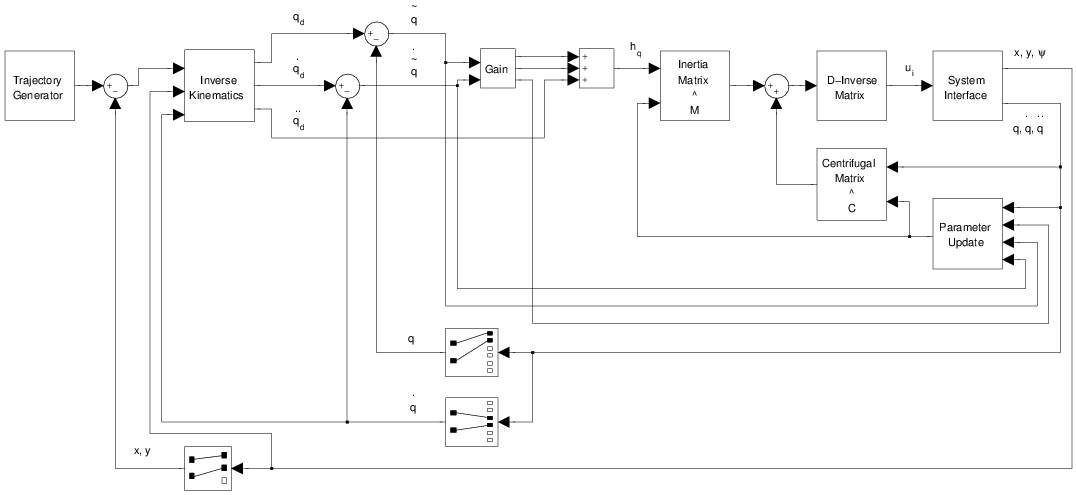


Fig. 7.13 Adaptive Tracking Control System of the WMR (Deployed Model)

7.4.1 Parameter Update (Deployed Model)

It was explained in the Subsection 7.3.7.3 that MATLAB's lyap() function could not be compiled for the WMR's microcontroller. Hence, a slight modification had to be made to the Parameter Update subsystem as shown in Fig 7.14. In place of the function for calculating the necessary symmetric positive definite constant matrix, a constant block (spd) is introduced. The values calculated in the simulation model are manually transferred into this constant block.

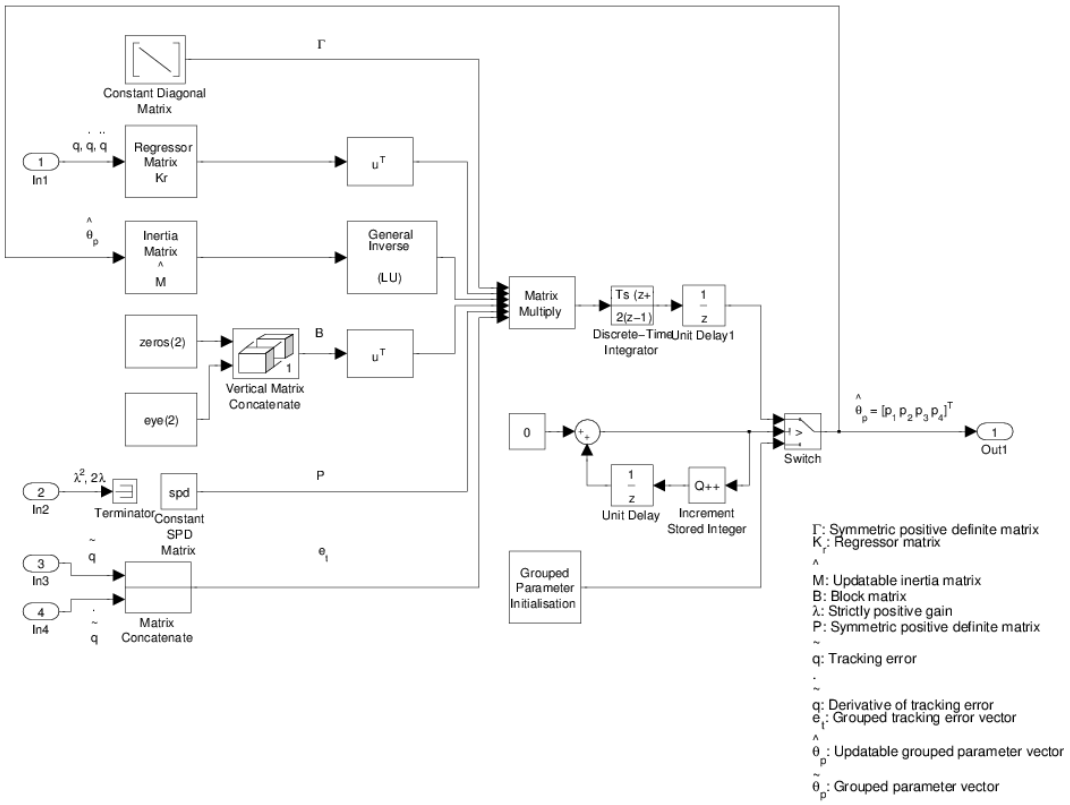


Fig. 7.14 Parameter Update Subsystem (Deployed Model)

7.4.2 System Interface

The System Interface subsystem as shown in Fig. 7.15 consists of interfaces with the DC motors and the optical mouse sensors. Conceptually, the input signals of the subsystem are the voltage levels required to drive the motors at the desired speeds. However, the motors are driven by PWM (pulse-width modulation) signals and not by direct voltage levels. Furthermore, the WMR's main microcontroller does not actually generate the voltage needed to power the motors. Hence, the voltage levels are only represented as numerical magnitudes in the signals processed by the microcontroller. These signals are converted into corresponding duty-cycle signals which are then transmitted to the motors. The interface blocks for the DC Motors are part of the Villanova University Lego Real Time Target (VU-LRT) library that is designed for use with the Lego Mindstorms NXT system and Simulink.

At the same time as the control signals are being sent to the motors, the WMR is receiving data from the optical mouse sensors. The raw data from the sensors are unpacked and repacked into usable data packages by the Pack function before being sent to the Sensor Processing subsystem for further processing. The embedded code for the Pack subsystem is placed in Appendix A.

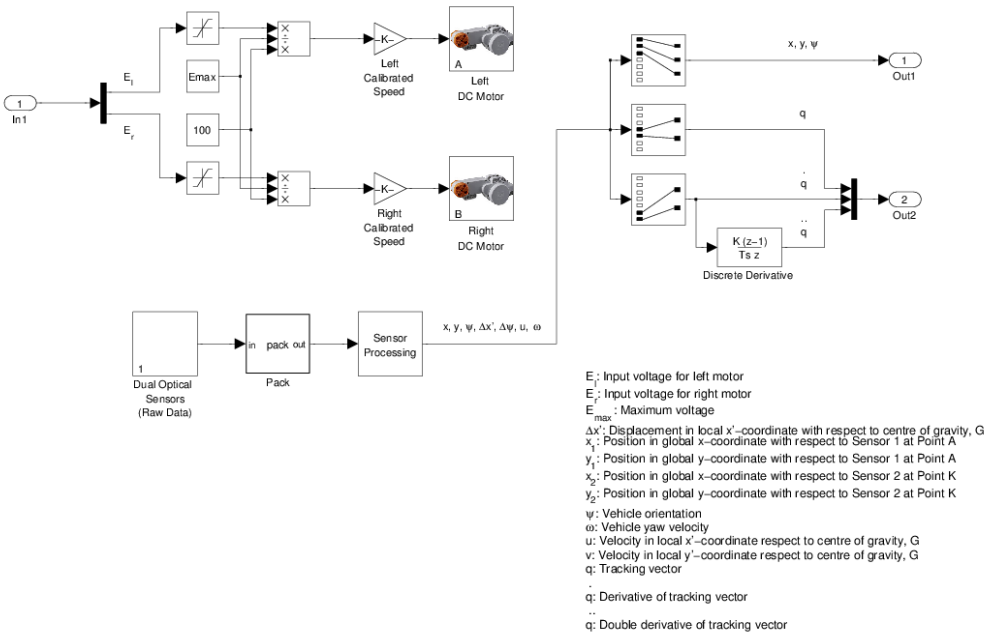


Fig. 7.15 System Interface Subsystem

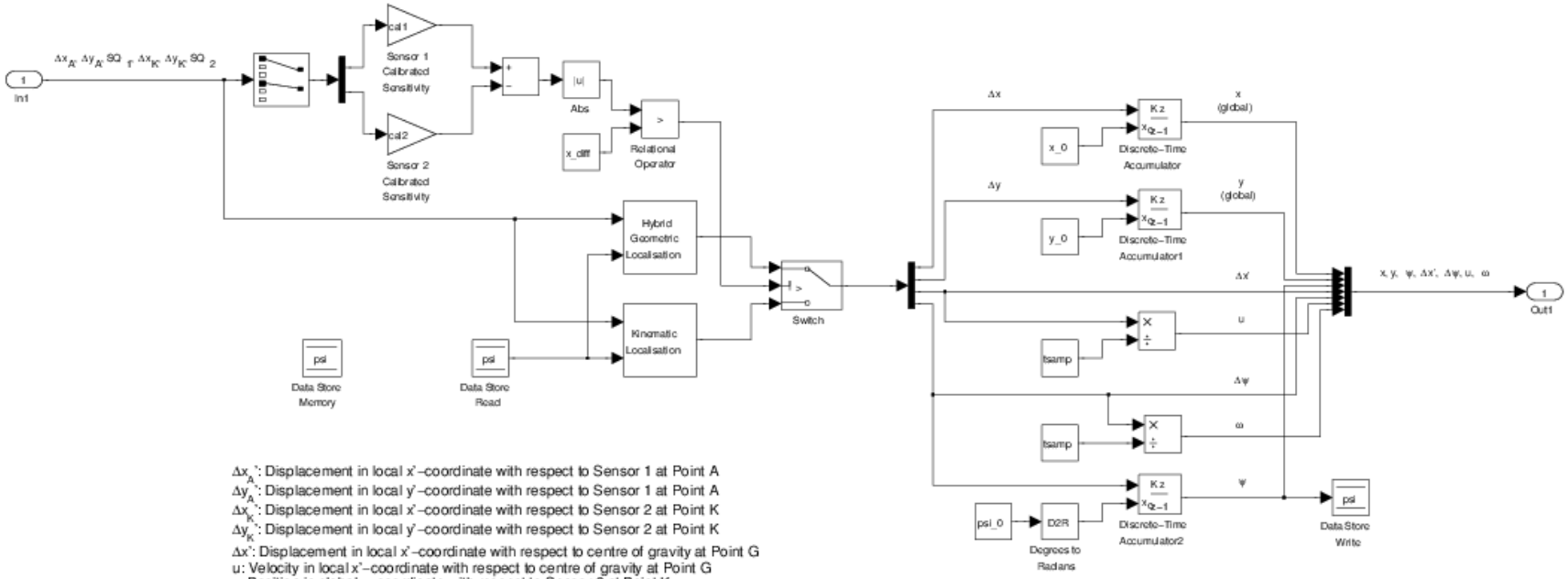
7.4.2.1 Dual Optical Sensors

The Dual Optical Sensors subsystem as depicted in Fig. 7.16 consists of a borrowed S-Function that forms the basis of several other sensor blocks within the VU-LRT library. Here, it has been adapted to interface with both optical mouse sensors.

Fig. 7.16 Dual Optical Sensors Subsystem

7.4.2.2 Sensor Processing

The Sensor Processing subsystem as shown in Fig. 7.17 is responsible for computing the sensor data to determine the displacement of the WMR during the last sampling period, and its current position with respect to the global coordinates. The reliability of the sensor data is first checked according to the method described in Section 6 of Chapter 6. If the data from both sensors are deemed acceptable, the Hybrid Geometric Localisation subsystem is activated. Otherwise, the Kinematic Localisation subsystem is used instead.



$\Delta x_A'$: Displacement in local x'–coordinate with respect to Sensor 1 at Point A
$\Delta y_A'$: Displacement in local y'–coordinate with respect to Sensor 1 at Point A
$\Delta x_K'$: Displacement in local x'–coordinate with respect to Sensor 2 at Point K
$\Delta y_K'$: Displacement in local y'–coordinate with respect to Sensor 2 at Point K
$\Delta x'$: Displacement in local x'–coordinate with respect to centre of gravity at Point G
u: Velocity in local x'–coordinate with respect to centre of gravity at Point G
x: Position in global x–coordinate with respect to Sensor 2 at Point K
y: Position in global y–coordinate with respect to Sensor 2 at Point K
$\Delta\psi$: Vehicle yaw
ψ: Vehicle orientation
ω: Vehicle yaw rate
x_0: Global x–coordinate of initial position
y_0: Global y–coordinate of initial position
psi_0: Global orientation of initial position
x_diff: Preset threshold of $|\Delta x_A - \Delta x_K|$
tsamp: Sampling time

Fig. 7.17 Sensor Processing Subsystem

7.4.2.3 Hybrid Geometric Localisation

There are two techniques used for odometric calculations as discussed in Chapter 6. Of the two, the hybrid geometric method offers better accuracy over the purely kinematic approach. Hence, the former method is employed for localisation as and when possible. However, it is not applicable to a single-sensor configuration. Although the WMR has two optical sensors, this scenario can arise when either of the sensors report faulty readings. This leaves only one set of reliable data and would necessitate a switch to the kinematic localisation method until the sensor readings are back to normal.

There are several embedded functions in the Hybrid Geometric Localisation subsystem as shown in Fig. 7.18. One of these functions sets the turning angle to zero once the calculated value dips below the minimum detectable turning (yaw) angle based on sensor sensitivity. This would evoke the special case scenario where the turning angle is perfectly zero. The other embedded functions change the geometric calculations to an Eulerian one when the critical radius has been reached. The codes of these embedded functions are placed in Appendix A.

7.4.2.4 Kinematic Localisation

The kinematic odometry method is based on a first-order Eulerian approximation technique and cannot match the accuracy of the geometric approach except when there is little or no turning. Hence, it would not be utilised unless it is necessary. This could happen when one of the sensors is producing erroneous readings. In that case, this is the back-up localisation method that would be used. The Kinematic Localisation subsystem is shown in Fig. 7.19.

When this subsystem is activated due to unreliable readings being detected, the algorithm will compare the reported SQUAL (surface quality) figures with calibrated values to see which of the sensors is most likely to be at fault. The sensor whose SQUAL readings fall within the acceptable range would have its data processed while the other's data would be discarded. If both sensors report poor SQUAL readings, the outcome would be highly questionable. This is because the system requires at least one set of reliable data from either sensor for odometric computation. In this case, data would be read from Sensor 1 by default. The alternative is to terminate the program and halt the WMR.

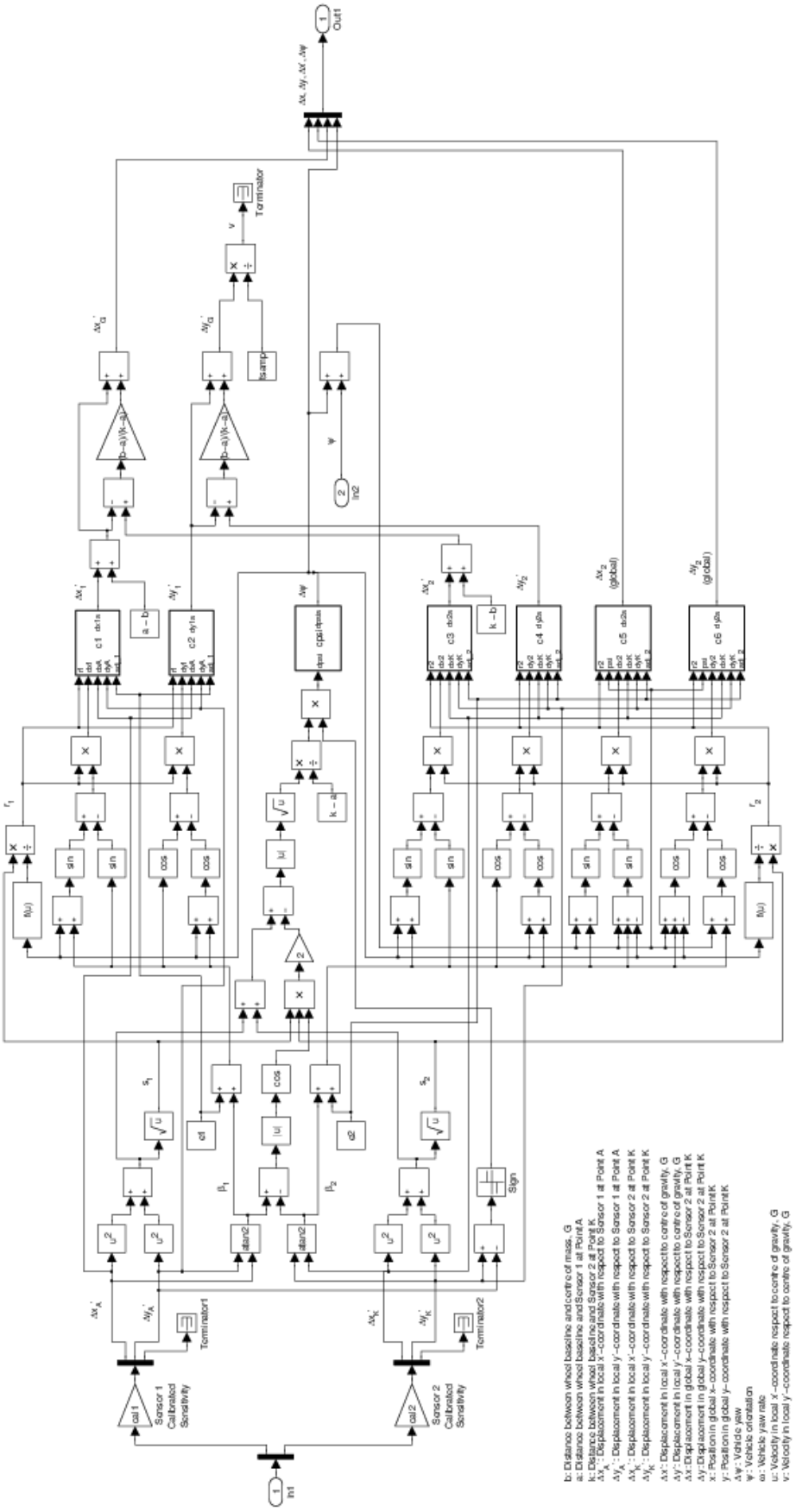
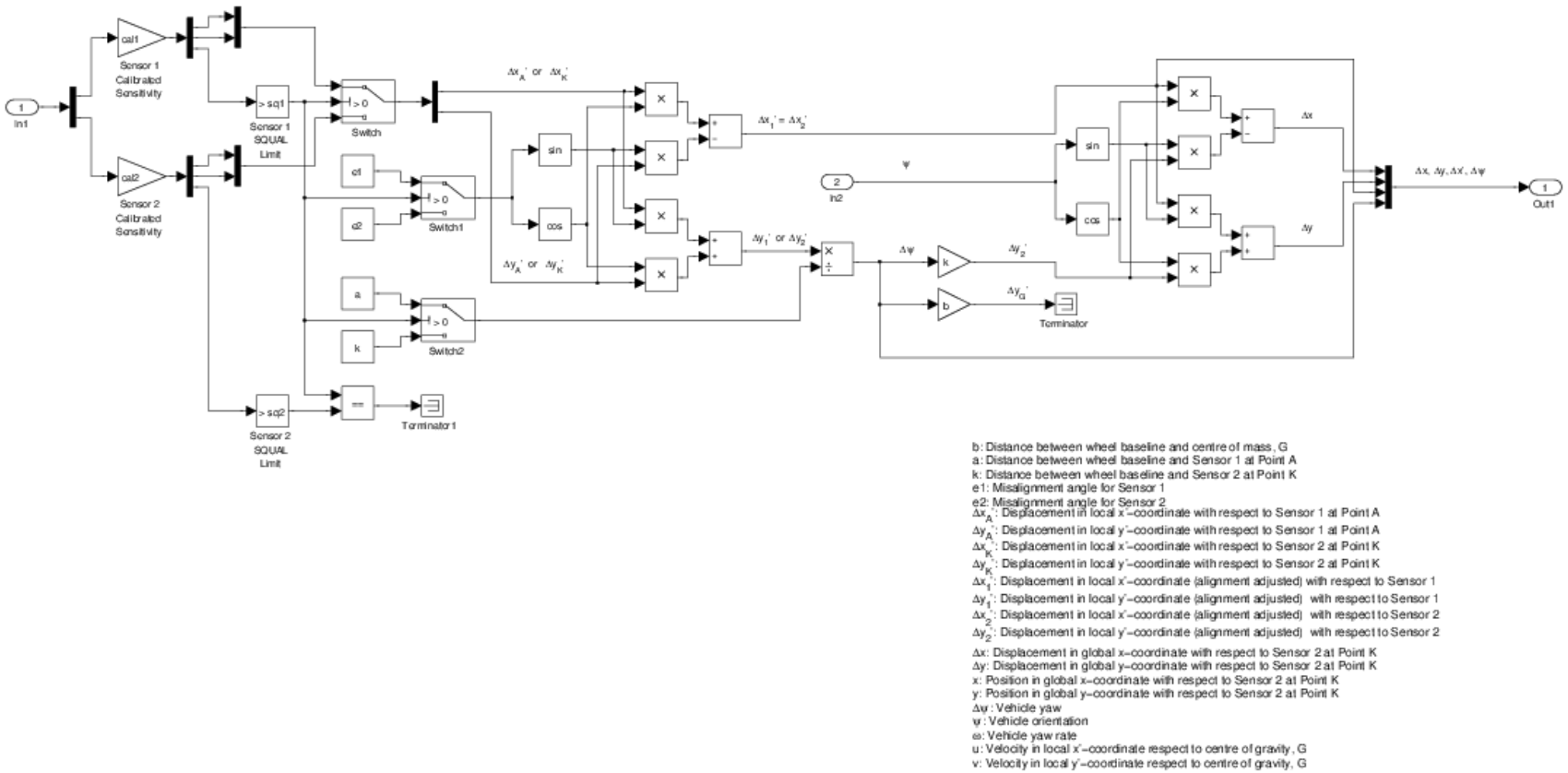


Fig. 7.18 Hybrid Geometric Localisation Subsystem



b: Distance between wheel baseline and centre of mass, G
a: Distance between wheel baseline and Sensor 1 at Point A
k: Distance between wheel baseline and Sensor 2 at Point K
e1: Misalignment angle for Sensor 1
e2: Misalignment angle for Sensor 2
$\Delta x_A'$: Displacement in local x'–coordinate with respect to Sensor 1 at Point A
$\Delta y_A'$: Displacement in local y'–coordinate with respect to Sensor 1 at Point A
$\Delta x_K'$: Displacement in local x'–coordinate with respect to Sensor 2 at Point K
$\Delta y_K'$: Displacement in local y'–coordinate with respect to Sensor 2 at Point K
$\Delta x_1'$: Displacement in local x'–coordinate (alignment adjusted) with respect to Sensor 1
$\Delta y_1'$: Displacement in local y'–coordinate (alignment adjusted) with respect to Sensor 1
$\Delta x_2'$: Displacement in local x'–coordinate (alignment adjusted) with respect to Sensor 2
$\Delta y_2'$: Displacement in local y'–coordinate (alignment adjusted) with respect to Sensor 2
Δx: Displacement in global x–coordinate with respect to Sensor 2 at Point K
Δy: Displacement in global y–coordinate with respect to Sensor 2 at Point K
x: Position in global x–coordinate with respect to Sensor 2 at Point K
y: Position in global y–coordinate with respect to Sensor 2 at Point K
Δψ: Vehicle yaw
ψ: Vehicle orientation
ω: Vehicle yaw rate
u: Velocity in local x'–coordinate respect to centre of gravity, G
v: Velocity in local y'–coordinate respect to centre of gravity, G

Fig. 7.19 Kinematic Localisation Subsystem

7.5 Summary

Based on the theoretical model proposed in the preceding chapter, a computational representation has been constructed here. Included in the model are all key components such as the adaptive controller, hybrid odometric system, partial-redundancy capabilities, motors and sensors. To be precise, there are actually two versions of this new computation model.

The first version is designed to run in a purely simulated environment. This will be a test for the soundness and stability of the model. It also allows the system's performance to be easily tweaked. Due to the nature of the simulation, the responses of actual hardware components, like the motors and sensors, had to be simulated.

Once the model's validity had been confirmed, a second version was developed. This latter version is meant to operate in a real-world scenario. It means the program is intended to be run on the CPU of the wheeled robot. As the motors and sensors no longer need to be simulated, the computational model had to be amended to replace the simulated components with hardware-interface modules. This sets the stage for the construction of the prototype.

Chapter 8: Test Prototype

8.1 Hardware Platform

Theoretical propositions can often be validated by carrying out real-world tests. Sometimes, observation will suffice if the hypothesis involves a natural phenomenon. Other times, a test prototype may have to be built in order to verify the soundness of a proposed idea.

One of the objectives of this research is to develop an accurate tracking system using low-cost components. Off-the-shelf parts are often (but not always) cheaper than custom-built devices. In this case, the Lego Mindstorms NXT robotics system was deemed a suitable candidate for the purpose of this study. It has a programmable embedded processor that comes along with plenty of components (i.e. construction parts, motors and sensors) that can be used to build a fully-functioning test platform. Furthermore, the standard Lego NXT kit is relatively inexpensive and has proven capabilities in robotics for hobbyists.

The optical tracking sensors used for this research are typically found on computer mice. Unfortunately, they are not available in a form where they can be plugged directly into the Lego robotics system without first undergoing some customisation. Also, since the optical sensors and the Lego NXT utilise very different communications protocols, an additional microcontroller has to be enlisted as an intermediary translation device.

8.1.1 Lego Mindstorms NXT

The Lego Mindstorms NXT 1.0 robotics system was first unveiled in July 2006. It was released as a successor platform to the highly popular Mindstorms Robotic Invention System (RIS). With an upgraded processor, servo motors with much more torque and a wider range of available sensors, the NXT is a much more advanced platform than its predecessor. Fig. 8.1 shows the NXT system connected to a maximum of three servo motors and a selection of available sensors.

Fig. 8.1 Lego Mindstorms NXT 1.0 robotics system (courtesy of Lego)

Not long after its introduction, Lego published the firmware code with an open source licence. Also made available to the public are the software developer kit (SDK), hardware developer kit (HDK) and schematics for all the standard hardware components (Lego 2006). Due to the open nature of the NXT system, it is now supported by well over a dozen programming languages and software development platforms in addition to the official NXT-G graphical programming environment. There are also plenty of third-party sensors available to complement those made Lego.

Thus, it is not surprising that since its debut six years ago, the Lego Mindstorms NXT has become widely embraced by the public, and is even used as a teaching tool (Grega and Pilat 2008; Bobtsov et al. 2011; Cuéllar and Pegalajar 2011; Cruz-Martín et al. 2012) for engineering courses in some universities. A minor revision (version 2.0) was released in August 2009 with upgraded software and sensors. However, the main processor block remained unchanged.

The physical block that contains the NXT's processing circuitry is called the Intelligent Brick. Within the Brick, the main processor is an Atmel AT91SAM7S256 CPU based on the 32-bit ARM7 microarchitecture. The processor runs at 48 MHz, and has 256 kB of flash memory and 64 kB of RAM onboard. It does not have a floating-point unit (FPU) and has to rely on emulation to handle floating-point numbers. For purposes that require a lot of floating-point operations, the lack of an FPU could severely affect the performance of the processor.

The ARM7 processor is widely deployed as an inexpensive and low-power microcontroller for embedded systems. This class of processor was first introduced in 1996, so its processing capabilities are anaemic in comparison to the latest generation of microcontrollers - not to mention microprocessors used in computers. Nevertheless, these processors continue to be used in current devices and have proven capabilities as demonstrated by the NXT.

The NXT is typically programmed to carry out simple to moderately complicated tasks. So, complex programs that also require plenty of floating-point operations may pose a challenge for the main processor. How well the NXT's processor fares will be revealed when it is finally put to the test.

In addition to the main processor, there is also an Atmel ATmega48 co-processor based on the 8-bit AVR microarchitecture. The ATmega48 runs at 8 MHz and has 4 kB of flash memory and 512 B or RAM. The main functions of the co-processor is to handle power management, perform analogue-to-digital (A/D) conversion of input signals, and create pulse width modulation (PWM) signals for the motors. An illustration of the NXT system architecture is shown in Fig. 8.2.

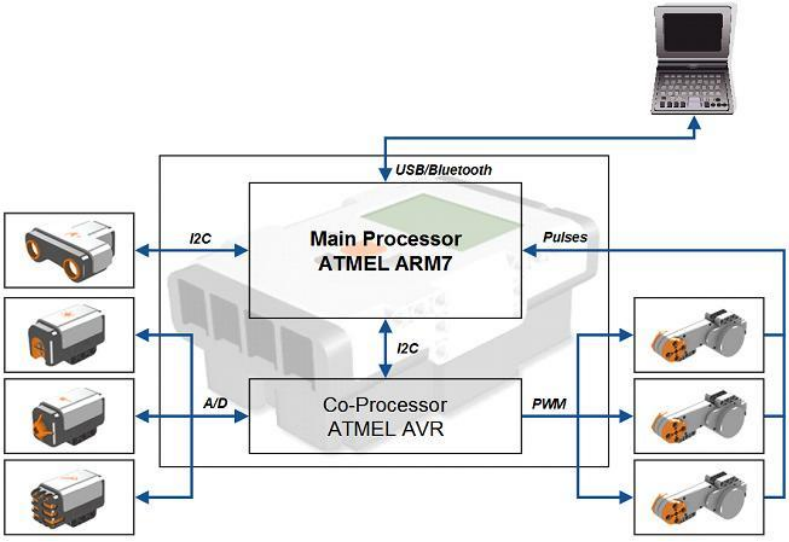


Fig. 8.2 Lego Mindstorms NXT system architecture (Chikamasa 2007)

The NXT can process multiple types of signals, such as PWM for the motors, digital I^2C (Inter-Integrated Circuit) and analogue signals from the sensors, USB 2.0, and Bluetooth. The wireless Blutetooth interface is a convenient means for data acquisition especially during testing and troubleshooting. However, it was discovered during initial tests that the maximum 9.6 kb/s transfer rate was not only rather slow but was further hampered by significant latency. Thus, the 12 Mb/s USB 2.0 connection is preferred over Bluetooth despite the necessity of using a cord.

The NXT can be powered by six AA rechargeable NiMH or non-rechargeable alkaline batteries for a nominal operating voltage of 7.2 V or 9 V. Alternatively, a rechargeable 7.2 V Li-ion battery pack can be used. The Li-ion battery pack is the preferred power option of this research.

8.1.1.1 NXT Device Communications

The main ARM7 processor has one hardware I^2C channel and uses it for communicating with the AVR co-processor via high-speed mode (380 kb/s). The design of the NXT system does not anticipate the data transmission rates of connected I^2C peripherals to come anywhere near the high-speed range. To prevent slow device communications from bogging down the only I^2C channel and affecting the transmission speed between the two processors, external I^2C device connections are offloaded onto a different port. However, since the ARM7 processor has only one native I^2C port, external I^2C communications have to be implemented using software emulation on a general purpose input/output (I/O) port. This is also known as a "bit-bang" approach. The factory-default maximum transmission rate of the low-speed I^2C port is 9.6 kb/s. This has since been boosted to 125 kb/s by third-party software developers (Shaw 2011).

The four input ports on the NXT into which sensors or other devices are plugged cater for both analogue and digital signals. Since the optical mouse sensor operates on a digital basis, this report shall therefore explore only the digital I^2C communication interface.

I^2C is a multi-master single-ended serial bus first introduced by Philips Electronics in the 1980s. The division (Philips Semiconductors) responsible for the design of I^2C was spun off in 2006 into a separate company called NXP Semiconductors. NXP waived licensing fees for the protocol soon after the company's creation, but continues to oversee the development of the technology. Some manufacturers refer to I^2C as TWI (two-wire interface).

There are only two bi-directional lines used for communications in the I^2C protocol in addition to power and ground lines. One is for the serial clock, usually labelled SCL, SCK or SCLK (DIGIAI0 on the NXT), and the other is for serial data, usually labelled SDA or SDIO (DIGIAI1 on the NXT). A generalised illustration of an I^2C system is shown in Fig. 8.3. On a typical I^2C bus, every connected device is assigned a unique 7-bit bus address and can assume either master or slave mode. The design of the NXT mandates that no other external device sharing the I^2C bus can assume the role of the master other than the NXT's main processor itself. This ensures that the NXT controls both the clock and signal flow.

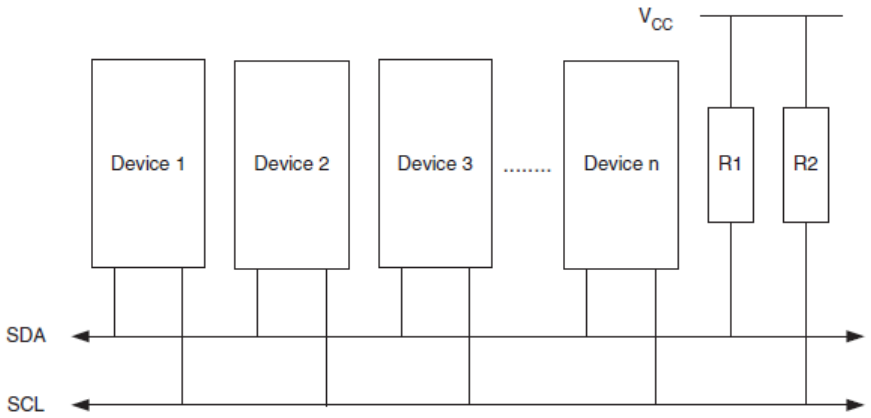


Fig. 8.3 I^2C bus interconnection (Atmel 2010)

In order to communicate with a slave device on the I^2C bus, the NXT master has to initiate a series of commands. For example, if the NXT is polling a particular sensor for data, the procedure would be as follows:

1. The NXT (master) issues a START command
2. If the target sensor (slave) uses internal registers to store its data, continue with Step 3, otherwise skip to Step 6
3. The NXT issues the 7-bit device (bus) address of the sensor plus a 1-bit WRITE flag
4. The NXT sends the 8-bit internal register address where the data is stored on the sensor
5. The NXT issues a STOP command followed by a repeated START command
6. The NXT issues the 7-bit device (bus) address of the sensor plus a 1-bit READ flag
7. The sensor transmits as much data as requested by the NXT for one polling cycle
8. The NXT issues a STOP command when it has received the required amount of data
9. The procedure is repeated for every polling cycle

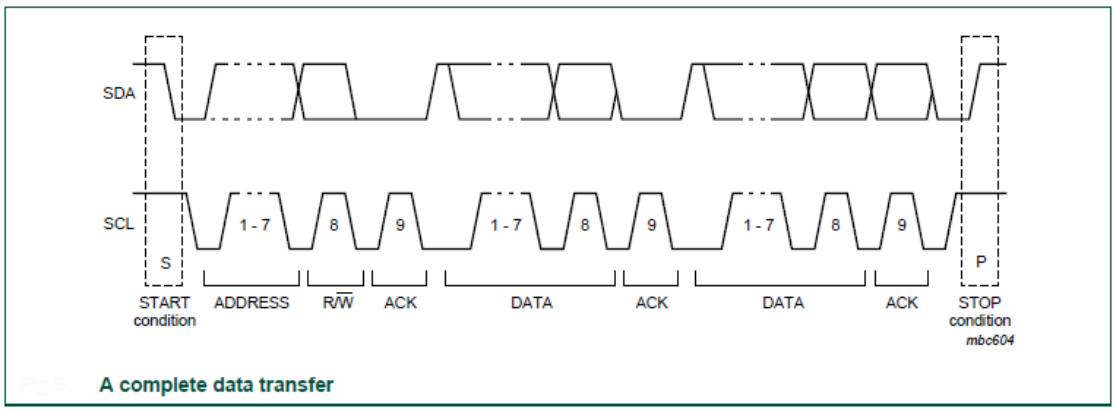


Fig. 8.4 I^2C data transmission (NXP 2012)

Note that both master and slave devices have to issue an acknowledgement at every step of the aforementioned procedure before the next action can take place. A graphical representation of a typical I^2C data transmission for devices that do not use internal registers is shown in Fig. 8.4.

Both the main and co-processors have native SPI (Serial Peripheral Interface) handling capabilities. However, the interface is not enabled on the NXT and no circuitry has been provided for it. Thus, it is somewhat of an inconvenience that the default communication protocol of the optical mouse sensor is SPI. The original microcontroller that accompanies the optical mouse sensor converts the signal from SPI to USB so that it can be used with modern computers. In this research, the sensor along with the laser diode and ceramic resonator are removed from the original circuit board and redeployed with custom-made circuitry that is connected to a different microcontroller. This SPI signal from the sensor is then translated into I^2C by the custom-programmed microcontroller so that sensor can communicate with the NXT.

8.1.1.2 NXT Direct Current Servo Motor

The NXT servo motor shown in Fig. 8.5 is the latest addition to the Mindstorms family of PWM-driven DC motors. It is the first to sport an encoder and has a one-degree resolution. This motor also delivers far more torque than any previous models in the product range. It powered by the output port of the NXT at 4.3V.

Fig. 8.5 Lego Mindstorms NXT servo motor (courtesy of Lego)

Lego has not released the technical specifications for the motor. However, numerous researchers and hobbyists have carried out their own testing and released their results. The most extensive analyses (Hurbain 2007a, b) have been conducted by the co-author of a prominent book on the Lego Mindstorms NXT (Gasperi and Hurbain 2009). Using the data provided by Hurbain, Prof Ryo Watanabe of Waseda University was able to calculate the motor characteristics (Watanabe 2007). Many have since used those parameters in their

projects (Yamamoto 2009a, b) and research. Some have also used Hurbain's data to complement their own results (Maxim 2011), while others have conducted their own experiments (Dalsager et al. 2006; Gonçalves et al. 2009; Martinec and Hurak 2011) to figure out the parameters of the NXT motor. There is quite a bit of variance in the results and it is not known whether the discrepancy in figures is due to different testing methodologies or manufacturing variability.

Taking the average of the published results of the NXT motor's electromechanical characteristics, the parameters used in this research is summarised in the table below:

Motor Parameter	Unit	Value
Torque constant, K_t	N.m/A	0.324
Back-EMF constant, K_b	V.s/rad	0.495
Armature resistance, R_a	Ω	5.263
Armature inductance, L_a	H	4.7e-3
Viscous-friction coefficient, B_m	N.m.s	6. 002e-4
Rotor inertia, J_m	$N.m.s^2$ or $kg.m^2$	1.321e-3

Table 8.1 Lego Mindstorms NXT motor characteristics

The values of torque constant (K_t) and back-EMF constant (K_b) in SI units should be identical in an ideal situation. However, due to unavoidable experimental error, this is clearly not the case. It is noted that some researchers only tested for the torque constant and presumed the back-EMF value to be of equal magnitude. Ultimately, the accuracy of these figures may not be of great importance due to the adaptive controller's ability to update the value of the WMR's parameters.

8.1.2 JED Microprocessors AVR200 Single Board Computer

The JED AVR200 is a single-board microcontroller with an Atmel ATmega32 processor at its heart. The board essentially hosts all the circuitry required to support the functioning of the microcontroller, such as providing power connections, clock generator, input/output (I/O) terminals, additional memory, etc. The AVR200 can be considered a self-contained processing system sans a power supply. A picture of the board and a diagram of its component layout are shown in Figs. 8.6 and 8.7 respectively. The full circuit schematics of the board are located in Appendix C.

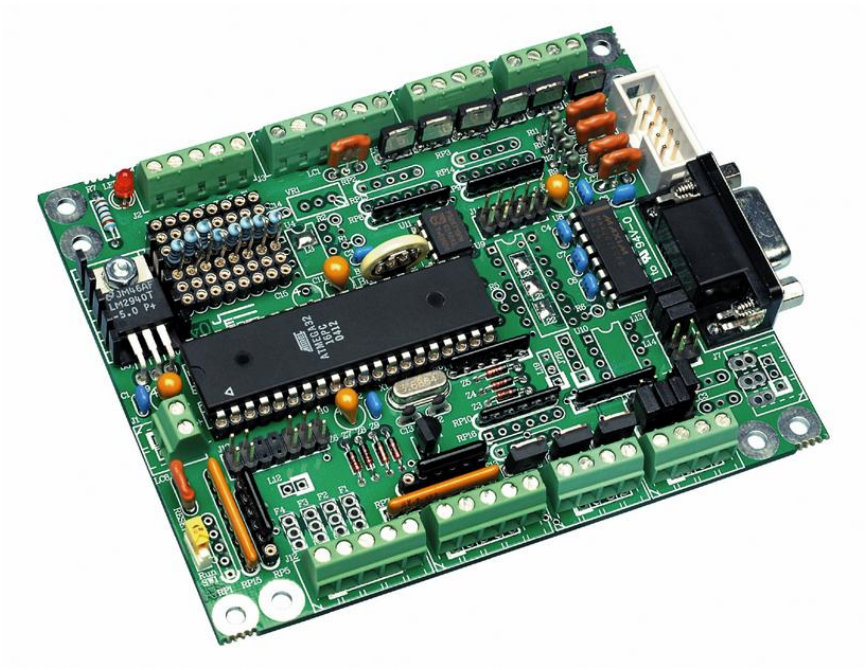


Fig. 8.6 JED AVR200 Single Board Computer (JED 2008)

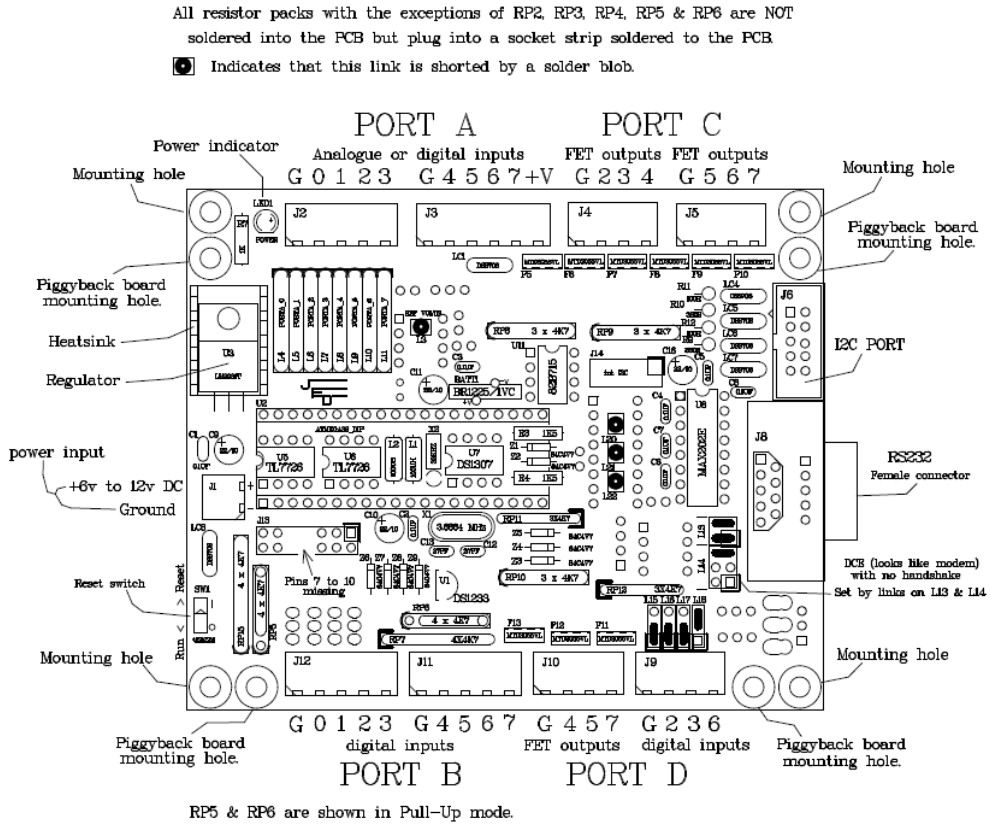


Fig. 8.7 JED AVR200 Single Board Computer PCB layout (JED 2008)

The ATmega32 microcontroller operates at 5 V while the AVR200 controller board accepts DC inputs between 5.5 V and 18 V (Schoell 2004). It is tolerant of transient voltage spikes ranging from -50 V to 60V. On the NXT, there are three different voltage output levels available at each port, namely, 3.3 V, 4.3 V and 9 V (7.2 V if using rechargeable batteries). There were initial attempts to use a 9 V output from an unused NXT port to power the AVR200 board together with the two optical mouse sensors albeit indirectly. The sensors would be powered through a regulated 5 V output terminal on the AVR200 board, but the voltage would have to be stepped down further to 3.3 V for the sensors. However, regular signal drop-outs suggested that the supply current was unstable at the required level or simply inadequate. Thus, it was decided that the AVR200 board and two sensors should employ separate power supplies.

After switching the power source for the microcontroller board and optical sensors, the use of an enclosure for six AA NiMH batteries was initially planned. However, it was discovered that the NiMH batteries could sustain an effective voltage of 1.45 V under load for a good length of time, even though their nominal voltage is rated at 1.2 V. This means that the only four batteries would be needed. Indeed, this has been verified during extended testing.

8.1.2.1 JED AVR200 Board Modifications

In order for the AVR200 board to work with the NXT, some minor alterations had to be made. Firstly, the AVR200 has a buffered I^2C bus via the use of a Philips 82B715 I^2C bus extender chip (JED 2005). The NXT does not have a similar chip within its I^2C circuitry, so 82B715 chip had to be desoldered and removed.

Secondly, the NXT has no internally-mounted pull-up resistors. Lego recommends the use of 82 kΩ resistors on both the clock and data lines of an external device (Lego 2006). On the AVR200, the clock and data lines each has a 330 Ω and 1.5 kΩ resistor connected to Vcc (JED 2005). They had to be removed and replaced with a single 82 kΩ resistor on each line.

8.1.2.2 Atmel AVR ATmega32 Microcontroller

The single most important component on a controller board is the microcontroller itself. Without it, the board has no reason to exist. On the AVR200 board sits the Atmel ATmega32 microcontroller. It is an 8-bit processor within the AVR microarchitecture family (Atmel 2011b). This particular microcontroller is regulated by an external crystal oscillator to operate at 3.6864 MHz, and has 32 kB of programmable flash memory, 1 MB of EEPROM (electrically-erasable programmable read-only memory) and 2 kB of SRAM (static random access

memory). It has native support for all the communication protocols required for the testing and operation of the prototype vehicle, such as SPI, I^2C (called TWI by Atmel) and USART (Universal Synchronous/Asynchronous Receiver/Transmitter).

The ATmega32 operates at 5 V. However, both the NXT's AT91SAM7S256 processor and the ADNS-6010 sensors operate at 3.3 V. The AT91SAM7S256 is tolerant of 5 V I/O signals (Atmel 2011a), so it can have direct signal lines with the ATmega32. However, the sensors are not 5V-tolerant, so the 5 V signals from the ATmega32 to ADNS-6010 have to be stepped down. The only signal that goes from the ADNS-6010 to ATmega32 is carried by the MISO line. The ADNS-6010 outputs a high signal within the range of 2.64 V to 3.3 V, while the ATmega32 registers a minimum of 3 V as a high signal. It is clear that the ATmega32 cannot detect the entire high signal range of the ADNS-6010. Therefore, a logic level shifter/converter was used to bring the 3.3 V signals to a 5 V level.

There are four main communication ports on the ATmega32, i.e. Ports A to D. In addition to handling general I/O functions, most port pins also have alternative uses. For example, Port B serves an alternative role as an SPI port, Port C doubles as an I^2C (TWI) port, and Port D also functions as a USART port.

The optical mouse sensors used in the prototype has a native SPI connection while the NXT uses I^2C instead. In order for the NXT and the sensors to communicate with one another, the signals have to be converted from I^2C to SPI. This is the purpose of the ATmega32 microcontroller.

8.1.2.3 Serial Peripheral Interface (SPI) on the ATmega32 in Master Mode

The Serial Peripheral Interface is a synchronous serial data communication protocol. It operates in full duplex, which means data can flow in both directions concurrently, unlike the half-duplex I^2C. Thus, it is generally a much faster interface than I^2C. As in I^2C, devices assume master and slave modes in SPI. However, no more than two devices can communicate with each other at any given time. Shown in Fig. 8.8 are timing diagrams of SPI transmissions reflecting the effects of clock phase polarity.

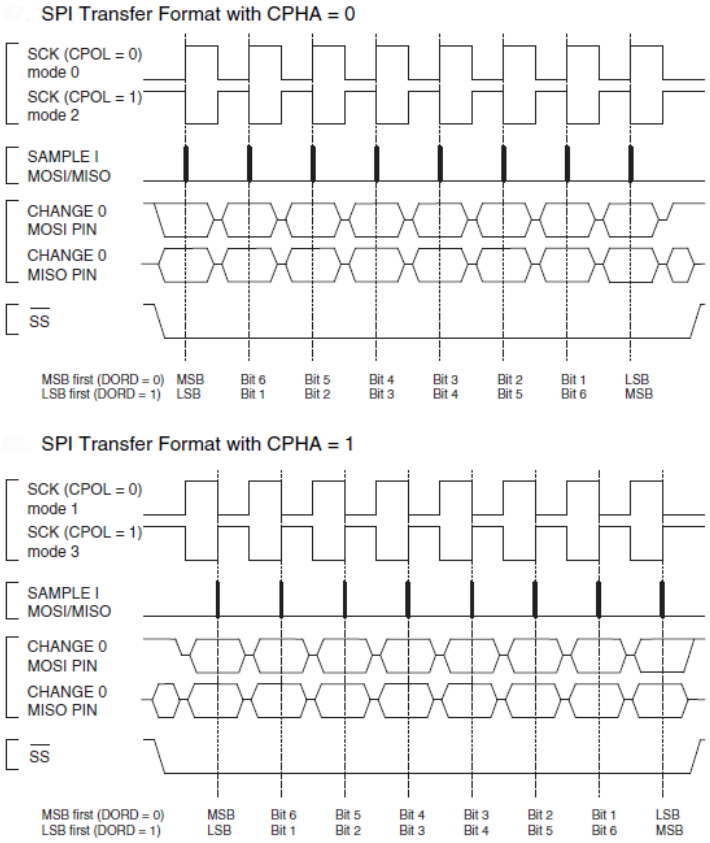


Fig. 8.8 SPI transmission with various clock phase and polarity settings (Atmel 2011b)

The ATmega32 acts as a master device (Atmel 2008) when communicating with the optical sensors. It sets the resolution of the sensors and also determines how often they are polled for data.

Aside from power and ground lines, the four main signal lines on the SPI bus are:

1. Serial clock (SCK, SCLK or CLK)
2. Master input, slave output (MISO)
3. Master output, slave input (MOSI)
4. Slave (or chip) select (SS, CS or NCS)

On the ATmega32, the aforementioned logic signals are assigned to Pins 7, 6, 5 and 4 respectively on Port B.

The main SPI configurations are found on the SPI Control Register (SPCR) (Atmel 2011b). The following is a brief description of their functions and respective settings on the ATmega32. Other settings are described in the microcontroller program itself.

1. Bit 7 - SPIE (SPI Interrupt Enable): Set to high for interrupt to be executed when a serial transfer is completed. **[Remains 0 by default]**
2. Bit 6 - SPE (SPI Enable): Set to high to enable SPI. **[Set to 1]**
3. Bit 5 - DORD (Data Order): Set to high to send LSB first and low to send MSB first. **[Remains 0 by default]**
4. Bit 4 - MSTR (Master/Slave Select): Set to high to configure AVR as master and low to configure it as slave. **[Set to 1]**
5. Bit 3 - CPOL (Clock Polarity): If set to high, SCK is high when idle; if set to low, SCK is low when idle. **[Set to 1]**
6. Bit 2 - CPHA (Clock Phase): Set to high to sample data on trailing edge and low to sample data on leading edge. **[Set to 1]**
7. Bits 1 and 0 - SPR1 and SPR0 (SPI Clock Rate Select): Set frequency of clock signal. **[SPR0 remains 0 and SPR0 set to 1]**

The settings for the clock polarity and phase are determined by operational characteristics of the optical mouse sensor (Avago 2009b). The SPI clock rate governs how fast the optical sensor is polled and is also configured to match the sensor's data update speed. The full program that is loaded on the microcontroller is found in Appendix B.

8.1.2.4 Managing Two SPI Devices on a Single ATmega32 SPI Port

When the NXT requests for data from the optical sensors, it expects the data to be transmitted from both sensors at the same time. However, there is only one native SPI channel on the ATmega32. So, it can only communicate with one SPI sensor at a time. This presents a significant problem.

It is possible to emulate the SPI function on another port. However, the emulated SPI would be far too slow to keep up with the hardware-based SPI. Other alternatives would be to acquire an additional identical microcontroller board or use a different type of microcontroller that has two native SPI ports, such as those in the Atmel AVR XMEGA family.

Ultimately, a solution to this problem was found that did not require any additional components. Both sensors would be connected to the same physical terminals of the SPI port (Port B) on the controller board except for the slave-select (SS) line. The SS wire of the first sensor would remain connected to the SS terminal of the SPI port. For the second sensor, the SS line would be routed to an unused pin on the same port and would be activated separately. Basically, the idea is to poll both sensors with a minuscule delay in between such that it would not have an adverse effect on the overall outcome.

When the sensors are polled for data, the SS line of the first sensor would be activated in order for the microcontroller to communicate with it and request for data. After the first sensor has been successfully polled, its SS line is set to low and data transmission will end. The procedure is then repeated for the second sensor.

When switching between sensors, there is naturally a delay. According to the hardware specifications of the sensors, there is a 120 ns delay between SS being set high and the SCK becoming active on a sensor, i.e. the sensor is ready for data transmission. The SPI polling rate is set by the microcontroller at 1/16th of the oscillator frequency which equates to 203.4 kHz. This means its sampling period is 4.34 µs. It is evident that the switching delay is extremely short compared to the sampling period. Effectively, it would appear to be almost instantaneous. Furthermore, each sensor has a data buffer that can store at least 16 data samples, so any slight delay or synchronisation issue would not have any noticeable effect on the data polling process.

8.1.2.5 Inter-Integrated Circuit (I^2C) Bus on the ATmega32 in Slave Mode

The Inter-Integrated Circuit bus is also called the Two-Wire Interface (TWI) by Atmel, the manufacturer of the ATmega32 microcontroller, as well as by many other chip makers. For this prototype, the ATmega32 receives data from the optical sensors via SPI, then repackages and transfers the same data to the NXT via I^2C. While the ATmega32 acts as an SPI master to the optical sensors, it functions as an I^2C slave (Atmel 2009) to the NXT. Thus, the NXT controls the data transfer rate between itself and the ATmega32. Details of its workings on the NXT are described in Section 8.1.1.1.

The I^2C portion of the program is interrupt-driven as opposed to the SPI section which is poll-based. Although the NXT software specifies the use of an internal register address for accessing data, the ATmega32 is programmed to ignore the command and send data from its default data register. Regarding the main I^2C configurations, they are found on the TWI Control Register (TWCR) (Atmel 2011b). The following is a brief description of their functions and respective settings on the ATmega32. Other settings are described in the microcontroller program as well as supplementary source and header files. The microcontroller is assigned a device/bus address of 0x01.

1. Bit 6 - TWEA (TWI Enable Acknowledge Bit): Set to high to enable ACK pulse when slave address or data is received. **[Set to 1]**
2. Bit 5 - TWSTA (TWI START Condition Bit): Set to high for device to become a master. **[Remains 0 by default]**

3. Bit 4 - TWSTO (TWI STOP Condition Bit): Set to high to generate STOP condition in master or for error recovery in slave. **[Remains 0 by default]**
4. Bit 2 - TWEN T(WI Enable Bit): Set to high to enable TWI operation and activate TWI interface. **[Set to 1]**
5. Bit 0 - TWIE (TWI Interrupt Enable): Set to high to enable interrupt request. **[Set to 1]**

The maximum I^2C transfer rate is limited to 9.6 kb/s (9.6 kHz) by the original operating system (OS) that came with the system. A few third-party software platforms for the NXT have been able to increase the I^2C speed to 125 kb/s. The prototype uses an open-source third-party OS that employs custom-designed driver software to manage the I^2C communications. Unfortunately, the drivers used by the open source OS is an older version that is still restricted to 9.6 kb/s. Newer versions of the driver provide a huge performance boost, but they are incompatible with the OS currently used by the prototype. Since the data sampling rate of the optical sensors far exceeds the maximum transfer speed of the NXT, data loss is a real risk. Indeed, at 203.4 kHz, the sensors sample at more than 21 times faster than the NXT is able to receive data. The problem is further compounded if the control program of the NXT samples at an even lower rate than is possible. A solution for this problem had to be found.

A study of the how data is stored and transmitted must first be conducted. An 8-bit variable can hold any number between 0 and 255 for a total of 256 possible values. A 16-bit (two-byte) variable can store numbers ranging from 0 to 65535 for a total of 65536 possible values. This means that a 16-bit variable could hold a minimum accumulation of 256 8-bit numbers. Naturally, the 16-bit variable can store more that 256 8-bit numbers if the individual 8-bit values are smaller than the maximum of 255. Based on how the data transfer process works, a 16-bit number can be sent in two 8-bit chunks. So, although a 16-bit variable can hold at least 256 8-bit numbers added together, it only requires twice the length of time to transmit compared to an 8-bit variable. Using this method, the rate of data transfer can be drastically reduced.

Each packet of sensor data corresponding to motion in the longitudinal and lateral directions is eight bits (i.e. one byte) long respectively. By creating a 16-bit buffer on the microcontroller for each set of coordinate data from the sensors, the vastly different data transfer rates between the sensors and NXT are no long an issue.

Through extensive troubleshooting, it was discovered that there was a flaw in the way that the I^2C driver communicated with the NXT. Data loss often occurred when any of the data bytes in a transmission held a value that exceeded 127. That number also happens to represent the maximum value that can be stored in seven bits. So, if the data bytes were instead cut up into 7-bit chunks (with a null eighth bit tacked on) before transmission, no data loss would occur.

However, this would mean that the 16-bit buffer variable would have to be transmitted in three parts.

In the end, it was unnecessary to transmit the buffer variable in three portions. During testing, the high sampling rate of the sensors and relatively slow speed of the prototype meant that value of each individual data packet was actually very small. The value held in the buffer never came close to exceeding a 9-bit maximum before being transmitted. So, the last two bits of the 16-bit buffer could be safely ignored and not transmitted. The code for the I^2C functionality of the microcontroller is arranged separately from the main program and is located in Appendix B.

8.1.2.6 USART on the ATmega32

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is a communications protocol that is used for transmitting data from the microcontroller to the computer during testing and troubleshooting phases. Although it is not used in the actual operation of the system, it played a vital role in getting the prototype up and running. The hardware connector for the USART port on the AVR200 board is of type RS-232/DE-9.

During the initial testing of the sensor and microcontroller, the involvement of the NXT was unnecessary. The first step of programming the microcontroller is to set up the SPI connection so that it can communicate with the sensors. To check whether the microcontroller has successfully acquired data from the sensors, the data on the microcontroller is transmitted via a serial cable to a computer for analysis. This data is transmitted through the USART port on the microcontroller and displayed on a terminal emulator program on a computer.

The USART code has been left intact in the main microcontroller program found in Appendix B and can be used for troubleshooting purposes when needed.

8.1.3 Avago ADNS-6010 LaserStream Laser Mouse Sensor

The Avago (formerly Agilent) ADNS-6010 was considered a high-performance sensor when it was introduced in 2005. Since then, optical mouse sensor technology has progressed significantly and delivery of the ADNS-6010 ceased in January 2012. The ADNS-6010 sensor is able to detect motion up to a speed of 45 in/s (1.14 m/s) and an acceleration of 20 g (Avago 2009b). It can process over 7080 frames (surface images) per second to provide high precision tracking. In addition, it has a selectable resolution/sensitivity of 400, 800, 1600 and

2000 counts/in (cpi) - equivalent respectively to 15.7, 31.5, 63.0, 78.7 counts/mm. The sensors used in the research were carefully desoldered from two brand-new mice. A picture of the sensor chip is shown in Fig. 8.9.

Fig. 8.9 Avago ADNS-6010 optical mouse sensor (Avago 2009b)

The operation of a typical optical mouse sensor is based on the principle of optical flow. The ADNS-6010 consists of two main components: the image acquisition system (IAS) and digital signal processor (DSP). The IAS comprises of a 716 μW laser diode, optical lens and image sensor. Essentially, the laser diode illuminates the tracking surface and the resulting image is reflected through the lens and captured by the 30-by-30-pixel image sensor. A continuous series of surface images (frames) are then processed by the DSP in order determine the displacements in the longitudinal and lateral directions with respect to the sensor. A cross-section diagram of the sensor, lens and laser diode is illustration in Fig. 8.10.

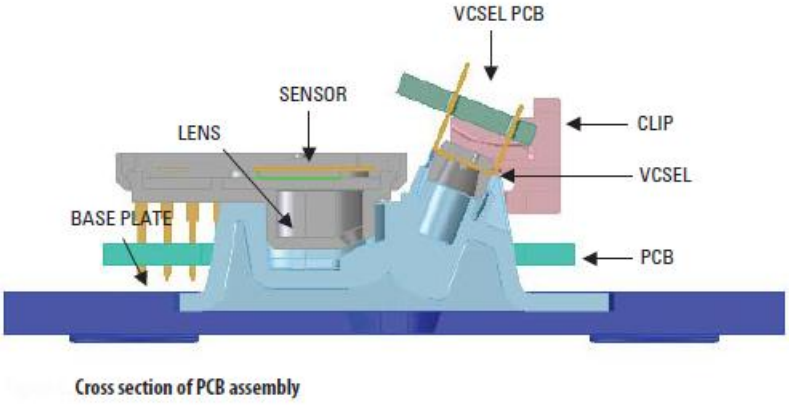


Fig. 8.10 Avago ADNS-6010 cross-section of sensor assembly (Avago 2009b)

The ADNS-6010 sensor operates at a 3.3 V level, so it has to use a voltage regulator to reduce the 5 V supply from the AVR200 board. For reasons unknown, the resolution of the sensors could not be set beyond 800 cpi. In any case, this is a very high level of resolution, so it is not a concern. The sensors assume the role of SPI slave devices while the ATmega32 microcontroller acts as the master. This means that the sensors' settings as well as their sampling rate are all dictated by the microcontroller.

8.1.3.1 Write and Read Operations on the ADNS-6010 Sensor

The ADNS-6010 sensor is the slave SPI device while the ATmega32 microcontroller takes on the role of the master. Thus, all read and write operations can only be initiated by the ATmega32.

A write operation is characterised by a transmission of data from the microcontroller to the sensor via the MOSI line. The first byte in every transmission sequence comprises of a 7-bit address plus an MSB (most significant bit) of "1". Data is contained in subsequent bytes. Data bits carried on the MOSI signal are sampled on rising edges of SCLK by the sensor. The timing diagram is illustrated in Fig. 8.11.

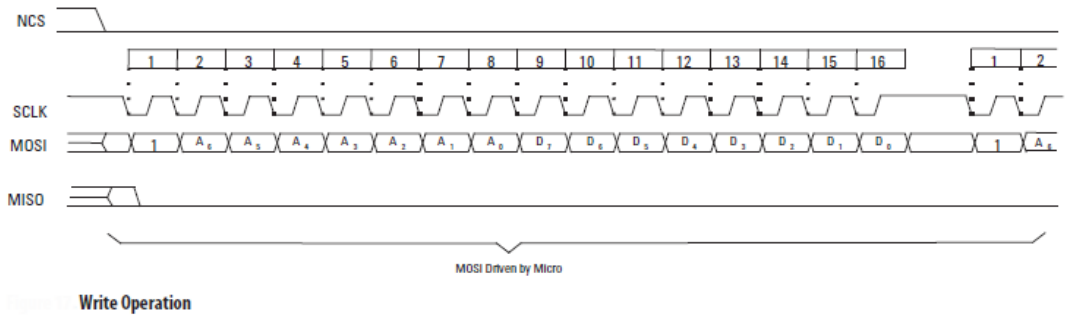


Fig. 8.11 Timing diagram for data sampling during write operations (Avago 2009b)

A read operation is characterised by a transmission of data from the sensor to the microcontroller via the MISO line. The first byte in every transmission sequence comprises of a 7-bit address plus an MSB (most significant bit) of "0". Data is contained in subsequent bytes. The sensor samples the MISO signal on rising edges of SCLK and outputs data bits on falling edges of SCLK. The timing diagram is illustrated in Fig. 8.12.

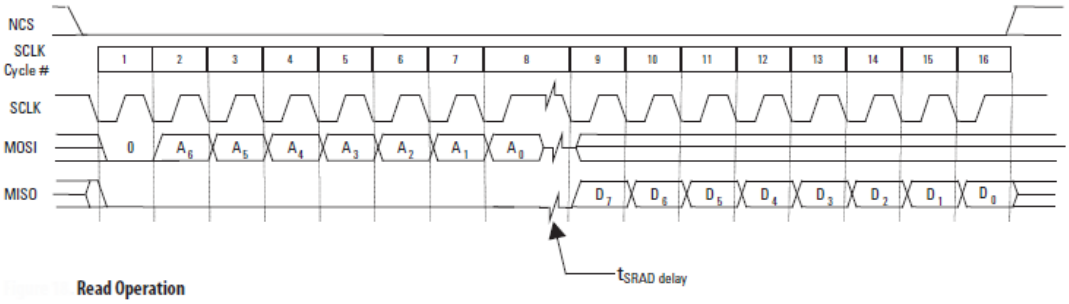


Fig. 8.12 Timing diagram for data sampling during read operations (Avago 2009b)

The timings of the data sampling for both write and read operations, as shown in Figs. 8.11 and 8.12, are used to configure the SPI clock phase and polarity on the ATmega32 microcontroller. In addition to the timings of the sampling process, the timings of write and read operations as well as any delays in between have to be fully taken into account when writing the microcontroller's program. These timings are shown in Fig. 8.13. The delays inserted into the program reflect this consideration.

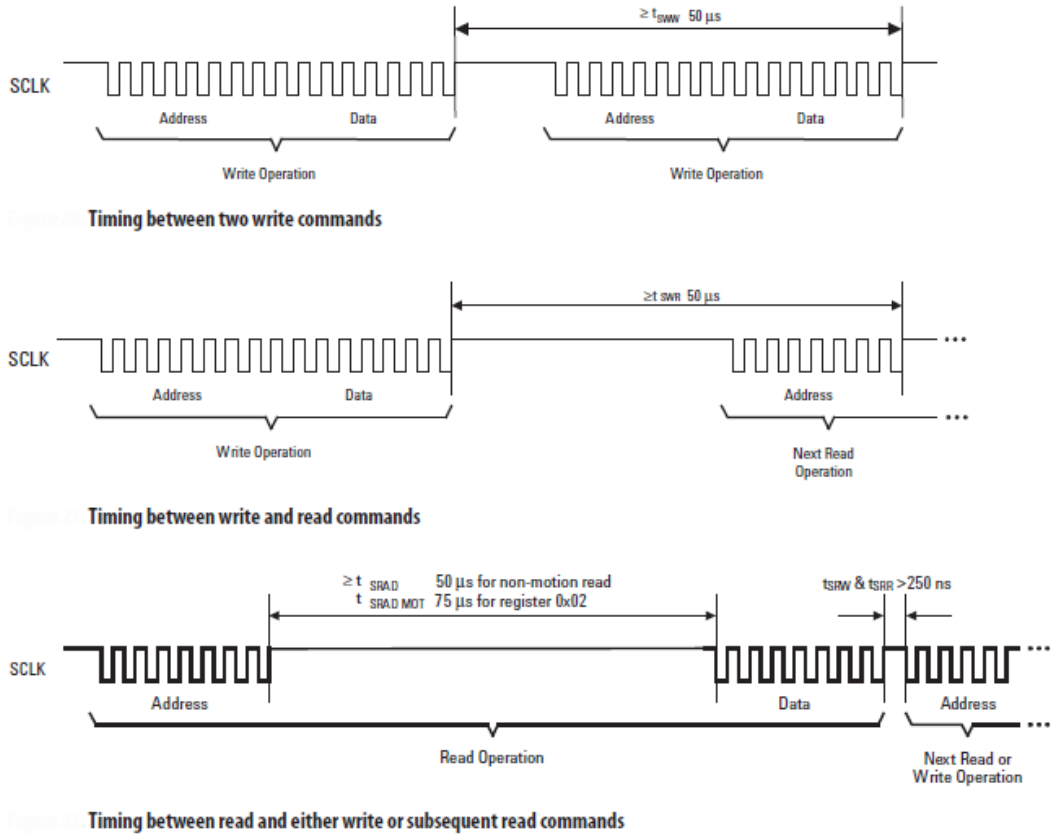


Fig. 8.13 Timing diagram for write and read operations and delays (Avago 2009b)

For the successful operation of the sensor, not every register has to be accessed. The following are addresses of 8-bit registers that have configuration bits which need to be set. This is accomplished by issuing relevant commands from the microcontroller to the sensor.

1. 0x0a - Configuration_bits: Sets resolution, laser mode, etc.
2. 0x2c - LP_CFG0: Sets relative laser current level
3. 0x2d - LP_CFG0: Complements 0x2C to set relative laser current level

For write and read operations in SPI, the following are addresses of 8-bit registers that are required to be accessed.

1. 0x02 - Motion: Reports any motion since last poll and any buffer overflows
2. 0x03 - Delta_X: Displacement in local X-axis (lateral axis of sensor) in counts
3. 0x04 - Delta_Y: Displacement in local Y-axis (longitudinal axis of sensor) in counts
4. 0x05 - SQUAL: Quality of surface image captured by sensor based on number of features detected

8.1.3.2 Sensitivity to Changes Between Sensor Assembly and Surface

The distance between the sensor assembly and tracking surface is a crucial factor in tracking accuracy. So, it is important to maintain this distance within operational limits. Naturally, a sensor with a greater operational range is better able to accommodate uneven surfaces, and thus less likely to lose tracking accuracy.

One of the improvements of the ADNS-6010 over sensors from an earlier generation is its greater operational range of distance between the sensor assembly and tracking surface. A good example is the popular ADNS-2051 model which predated the ADNS-6010 by about three years, and was a contemporaneous model with the latter in manufacturer's product range for several years.

Not only was the ADNS-2051 deployed in a wide array of retail mice, it has also been used in quite a few research endeavours as well as projects published on the World Wide Web. As is evident from their respective performance characteristics shown in Figs. 8.14 and 8.15, the ADNS-6010 is significantly more tolerant to a change in height of the gap between the sensor assembly and tracking surface than the ADNS-2051. The ADNS-6010 has a recommended operating region of 2.4 ± 0.3 mm and a maximum effective range extending to ± 0.8 mm. The ADNS-2051 has a similar recommended operating region to the ADNS-6010. However, its maximum effective range only extends to barely ± 0.4 mm before the onset of a marked deterioration of accuracy. This is one of the key factors influencing the selection of the ADNS-6010 for this research.

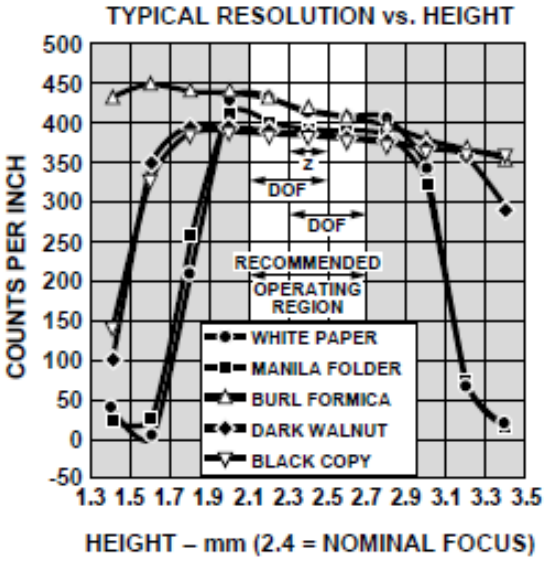


Fig. 8.14 Recommended sensor height for ADNS-2051 (Avago 2009a)

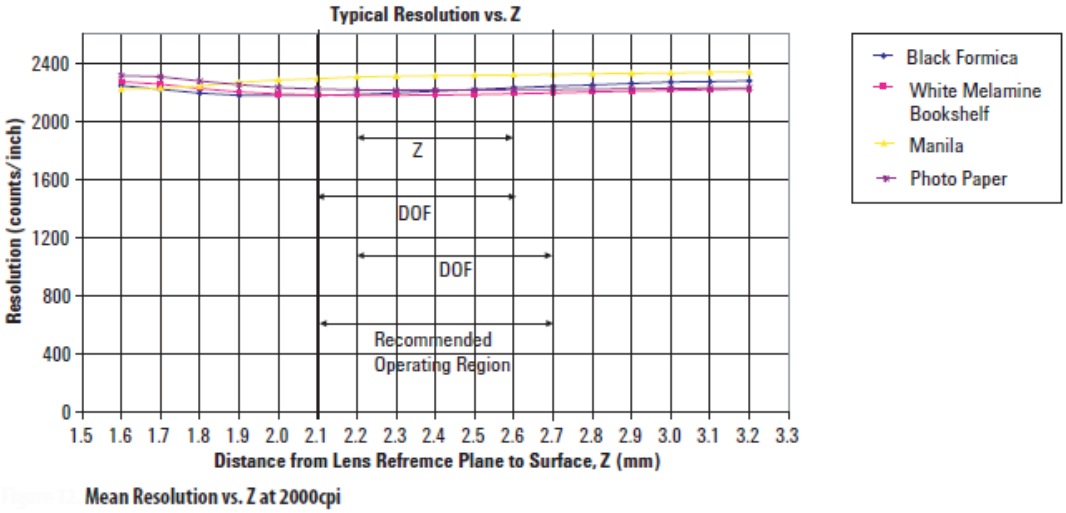


Fig. 8.15 Recommended sensor height for ADNS-6010 (Avago 2009b)

8.1.3.3 Construction of the ADNS-6010 Dual-Sensor Subsystem

The circuitry of the sensor is based on the documentation provided by the manufacturer. However, alterations had to be made because of the use of a different type of microcontroller. Also, extraneous components related to the buttons, scroll wheel and USB port were eliminated.

The sensors operate at 3.3 V but receive 5 V power from the AVR200 board. Thus, a voltage regulator had to be used to reduce the voltage to a suitable level. Similarly, the 5 V signals from the ATmega32 microcontroller had to be reduced. This was accomplished by the use of resistors and the principle of voltage division. For signals going from the ADNS-6010 to ATmega32, a logic level shifter/converter was used on the MISO line to step up the 3.3 V signal to a 5 V level. The level shifter unit also came with voltage-divider circuitry to step down 5 V signals to 3.3 V levels. However, the resistors used did not have an optimum ratio and halved the 5 V level instead. Thus, the voltage-divider circuitry provided by the level shifter board was not used.

According to the schematics of the AVR200 board, each of the four signal lines of the SPI port has a 4.7 kΩ resistor connected in series. The principle of voltage division states that:

$$V_{out} = \frac{R_2}{R_1 + R_2} V_{in} \tag{8.1}$$

Thus, the required resistance to step 5 V down to 3.3 V is:

$$R_2 = \frac{R_1}{\frac{V_{in}}{V_{out}} - 1} \tag{8.2}$$

$$R_2 = \frac{4.7 \times 10^3}{\frac{5}{3.3} - 1}$$

$$R_2 = 9.12\ k\Omega$$

Each sensor and all its supporting components and wiring are then mounted on a blank printed circuit board (PCB). The two sensor circuit boards are in turn mounted on a polycarbonate plate that has a pair of precision-cut holes for the lens assembly of the sensors. The resulting circuit diagram for the sensor is illustrated in Fig. 8.16. The fully-assembled sensor circuitry and mounting plate is shown in Figs. 8.17 and 8.18.

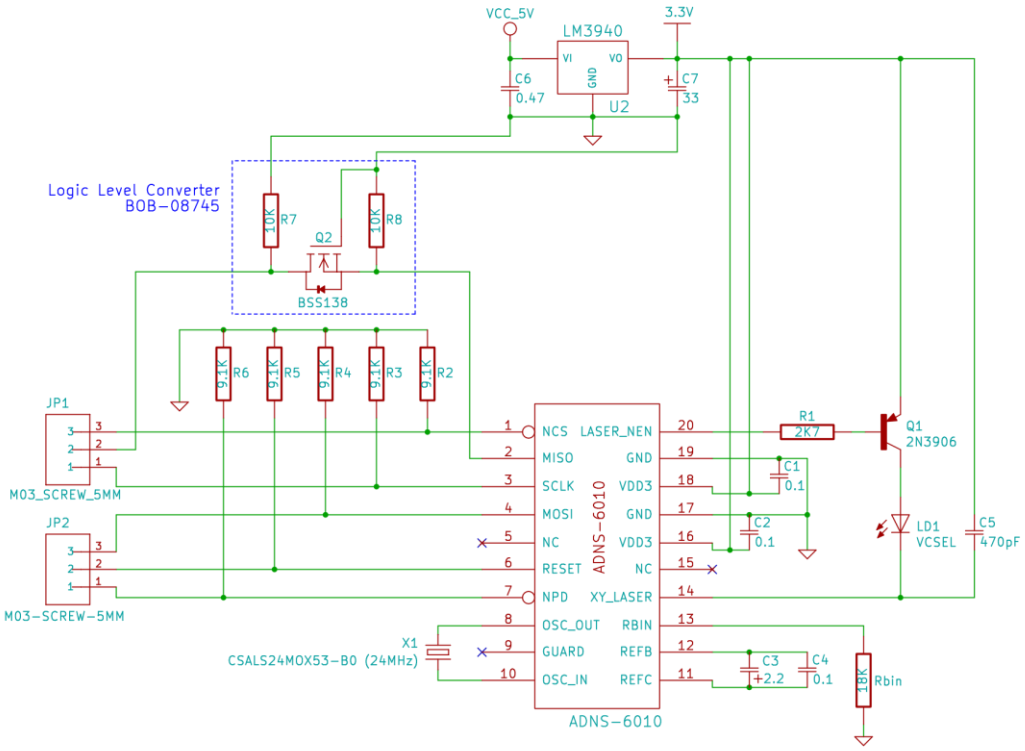


Fig. 8.16 Circuit schematic of ADNS-6010 optical mouse sensor subsystem

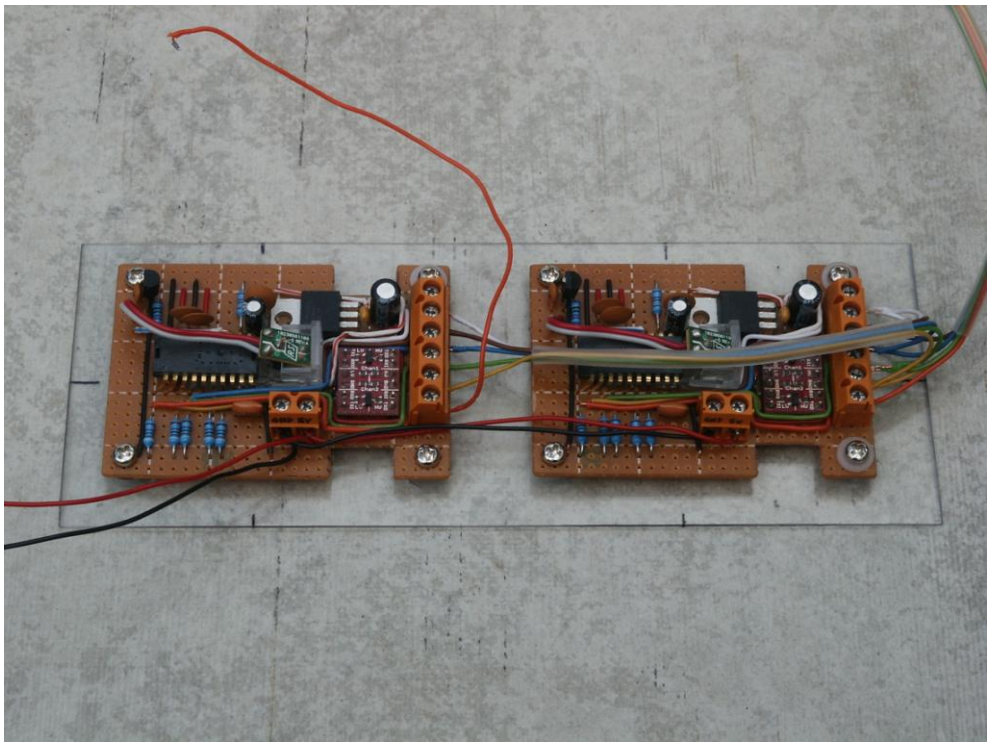


Fig. 8.17 Two optical sensors mounted on a polycarbonate plate

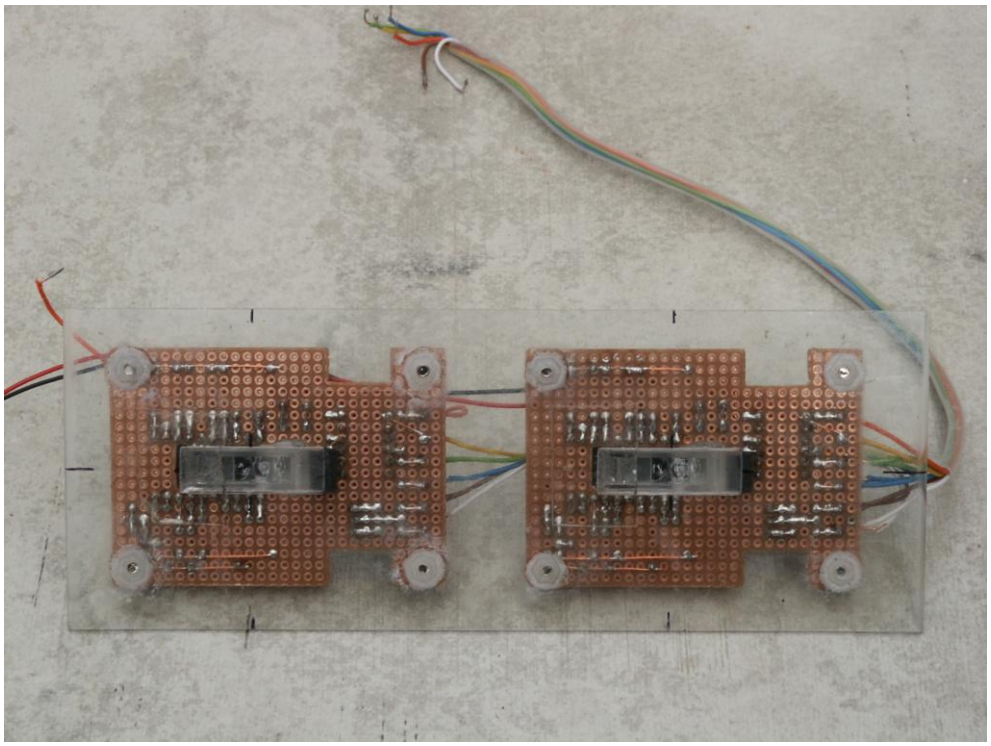


Fig. 8.18 Underside of sensor mounting plate

8.1.3.4 Calibration of the ADNS-6010 Sensors

Before an optical mouse sensor can be used for tracking, it has to be calibrated first. The displacement data reported by the sensor is measured in counts. Without a known correlation with actual measurement units, the data is quite useless. The selected resolution of 800 counts-per-inch (cpi) cannot be relied upon because it is only a nominal figure.

The first step is to affix the sensors along with their mounting plate to the prototype vehicle. The gap between the sensor plate and tracking surface was measured to be between 2 mm and 3 mm. There was not a practical method available to measure the distance to a precision of 0.1 mm. However, according to the hardware specifications as shown in Fig 8.15, there is no problem if the gap is within the range of 2 mm to 3 mm. This measurement was determined by the use of small steel plates of 2 mm and 3 mm thickness. The thinner plate was able to just slide through the gap, but thicker one was not.

As the vehicle's motors are geared, it is quite difficult to push it smoothly along a test surface. It is much easier to slide the vehicle along if there is a thin film under the tyres. A 0.13 mm plastic transparency was deemed to be a suitable material because it is very thin and also durable enough withstand the sliding motion for some time. To hold the vehicle in place while

it sits on the transparency, a wooden frame was constructed with steel corner braces, and aligned with the aid of a square ruler. The transparency was stapled across the frame. Holes were cut out in alignment with the sensors.

A test surface was found that had a straight steel guide and a distance of 2 m was marked out on the guide. For the calibration test, the frame with the vehicle firmly held within was slid along the guide from starting to ending markers. This process was repeated 10 times.

When the calibration runs were conducted, data had to be collected. The raw data from the sensors was first uploaded to the NXT and re-transmitted to a laptop and recorded. The distance for each run was divided by the total number of reported counts. The resulting number represents the sensor's sensitivity and has a unit of m/count, or alternatively mm/count. The data was averaged over the 10 runs. After calibration, when the sensor reports a displacement of a certain number of counts, the figure can be multiplied by the sensitivity factor to give the actual distance travelled in metres or millimetres.

The calibration results are shown in Tables 8.2 to 8.4. Ideally, the readings in the lateral axis (x-axis) of the sensor should be zero since there is no motion in that direction. However, it is impossible to achieve perfect alignment. Thus, a small amount of error is present, which has to be factored into the calculations. The results indicate that the resolutions for the two sensors are approximately 884 cpi and 834 cpi. Part of the calibration process includes a survey of the surface texture as seen by the sensors. This is called "surface quality" or SQUAL by the manufacturer. Also, a comparison of displacement readings from the two sensors was conducted to see how well they agree with each other.

	Sensor 1 X (count)	**Sensor 1 Y (count)**	**Sensor 2 X (count)**	**Sensor 2 Y (count)**
	16	68920	11	64690
	-86	70226	-56	66211
	-17	69818	42	65670
	16	69926	0	66030
	-77	70485	8	66390
	-55	69738	-91	65838
	-83	69480	-81	65506
	-57	69344	-21	65502
	-95	68750	-47	65196
	-14	69648	-32	65754
Average	-45.2	69633.5	-26.7	65678.7

Table 8.2 Sensor calibration data

	Sensor 1	Sensor 2
Sensor sensitivity (m/count)	2.872180E-05	3.045127E-05
Sensor sensitivity (mm/count)	2.872180E-02	3.045127E-02
Sensor offset angle (rad)	-6.491128E-04	-4.065245E-04
Sensor offset angle (deg)	-0.037191422	-0.023292136

Table 8.3 Calibrated sensitivity of both sensors

	Sensor 1 SQUAL	Sensor 2 SQUAL	Mean (x1 - x2) [m]	σ (x1 - x2) (using zero-adjusted mean) [m]	3σ [m]
	1174.238	1097.423	2.806e-04	3.079e-04	9.238e-04
	1166.833	1116.491	2.707e-04	3.264e-04	9.790e-04
	1170.773	1097.415	2.789e-04	3.302e-04	9.904e-04
	1167.350	1111.079	2.462e-04	2.795e-04	8.384e-04
	1168.817	1110.022	2.619e-04	3.079e-04	9.236e-04
	1163.838	1103.257	2.552e-04	2.947e-04	8.895e-04
	1163.670	1109.751	2.710e-04	2.992e-04	8.976e-04
	1166.011	1100.473	2.372e-04	2.785e-04	8.356e-04
	1165.600	1110.225	2.150e-04	2.632e-04	7.896e-04
	1164.053	1102.389	2.557e-04	3.156e-04	9.469e-04
Average	1167.118	1105.852	2.572e-04	3.003e-04	9.014e-04
σ	3.377	6.516			
3 σ	10.131	19.548			

Table 8.4 Data correlation and surface quality of test surface

8.1.4 Construction of Vehicular Prototype

The vehicular prototype was a differentially-driven wheeled robot with a single front castor wheel. Other than the microcontroller board and sensor subsystem, the chassis of the vehicle was entirely composed of parts from the Lego Mindstorms NXT 1.0 kit. The versatility of the NXT system meant that no custom manufacturing was required to build the vehicle chassis. Furthermore, the NXT kit was relatively inexpensive compared to most other commercially-available robotic platforms. This translated into savings in time and cost. An additional set

was purchased at the same time in case spare parts were needed. Six pieces from the second set were included in the current vehicle.

The overall dimensions of the vehicle were partly limited by the size of the individual parts. A larger vehicle may require more joints and result in a structurally weaker frame. If the vehicle were too small, it may not be able to contain the microcontroller board and sensor subsystem. While the microcontroller board was of a fixed size, the sensor subsystem and vehicle chassis were not. So, there was some flexibility in deciding on their final dimensions. The custom construction of the sensor subsystem and vehicle chassis also meant that they had to be designed somewhat concurrently and with the other in mind.

The design of the vehicle was quite straightforward. It had to conform to the design of a vehicle with differentially-driven rear wheels and a lone free spinning front castor wheel. Also, it had to be able to hold the NXT brick as well as additional components such as the microcontroller board and sensor subsystem. A final design was produced by adhering to these simple requirements.

Firstly, the two servo motors that drove the rear wheels were integrated as part of the structural frame. The gap between the two motors formed a channel that would fit the sensor subsystem. As the frame was built, space was allocated for the microcontroller board and also the NXT brick. As the chassis consisted mostly of beams (called lift arms by Lego) that were held together by connector pins, there was quite a bit of flexibility in the frame. Thus, in order to stiffen the frame, numerous diagonal pieces were added for bracing purposes.

The complete prototype is shown in Fig. 8.19. The vehicle in various stages of assembly is shown in Figs. 8.20 to 8.25. The final dimensions of the vehicle are displayed in Table 8.5. To calculate the mass moment of inertia, the vehicle was approximated to a rectangular box with the rotational axis situated at the rear end.

8.1.5 Measurement of Sensor Position on Tracking Surface

The designated sensor that defined the WMR's position was the front sensor, so it was only necessary to mark the position of this sensor and not the one in the rear. However, since the ground clearance between sensors and tracking surface was tiny, it was quite impossible to mark the position of the sensor directly on the ground. The solution was to form lines that intersect at the location of sensor on the vehicle and extend them to the periphery. The end points of these intersecting lines would be clearly marked on the vehicle as shown in Figs. 8.20, 8.21, 8.22 and 8.25. A vertical piece at the fore of the vehicle as well as outriggers on

both sides were added solely for this purpose. The ground clearance of outriggers was well under 1 mm and could barely fit a piece of paper in between.

In order to situate the vehicle at a starting location with a certain orientation, two lines that intersect the starting point were first drawn on the ground and extended just beyond the maximum dimensions of the vehicle. The vehicle was then moved into place by aligning its markings with these lines. Similarly, in order to determine the ending position, dots were drawn on the ground where the vehicle markings were located. The vehicle was then removed and intersecting lines were drawn by connecting the dots. The intersecting point would be the position of the ending point as identified by the front sensor.

Overall length (including rear wheels) [m]	0.262
Overall width (including outriggers) [m]	0.182
Height (including NXT brick) [m]	0.118
Weight [kg]	1.097
Track width [m]	0.126
Radius of tyre [m]	0.028
Distance between wheel baseline and first (rear) sensor [m]	0.045
Distance between wheel baseline and second (front) sensor [m]	0.135
Centre of gravity (longitudinal balance point) from wheel baseline [m]	0.102
Approximate mass moment of inertia [$kg.m^2$]	0.02

Table 8.5 Dimensions of wheeled robot

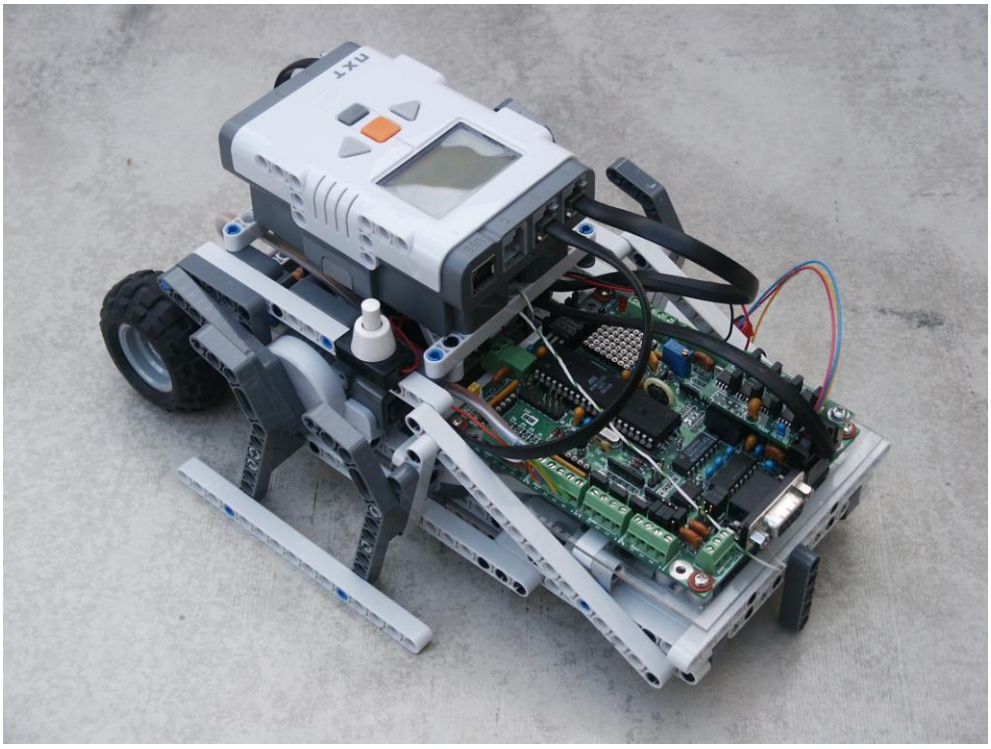


Fig. 8.19 Perspective view of fully-assembled vehicle

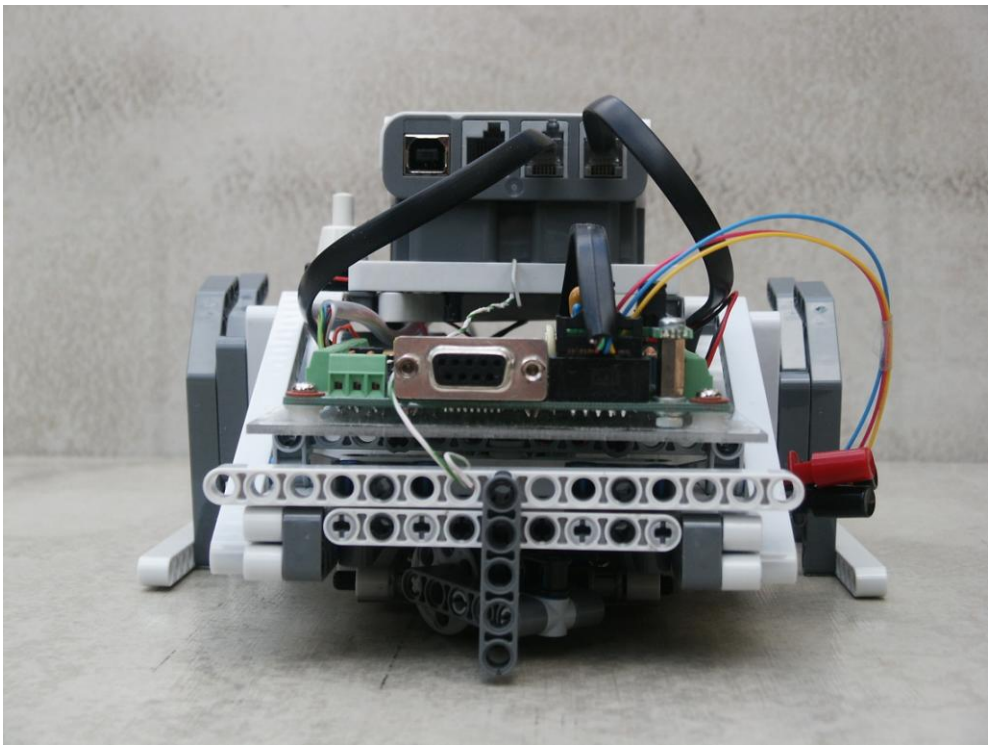


Fig. 8.20 Front view of fully-assembled vehicle

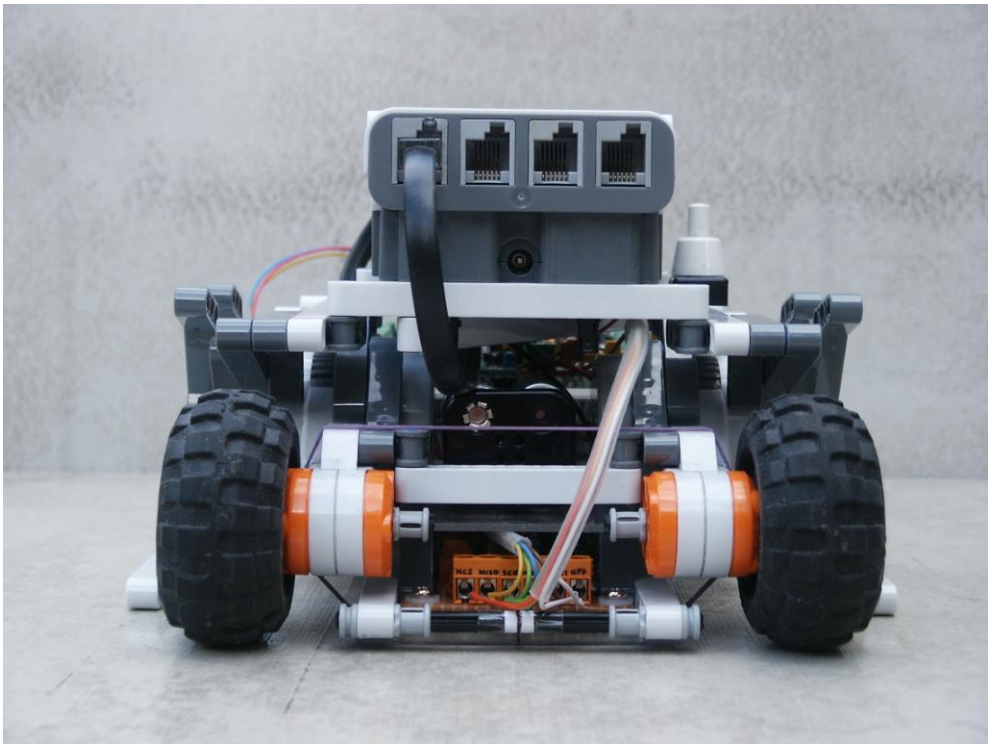


Fig. 8.21 Rear view of fully-assembled vehicle

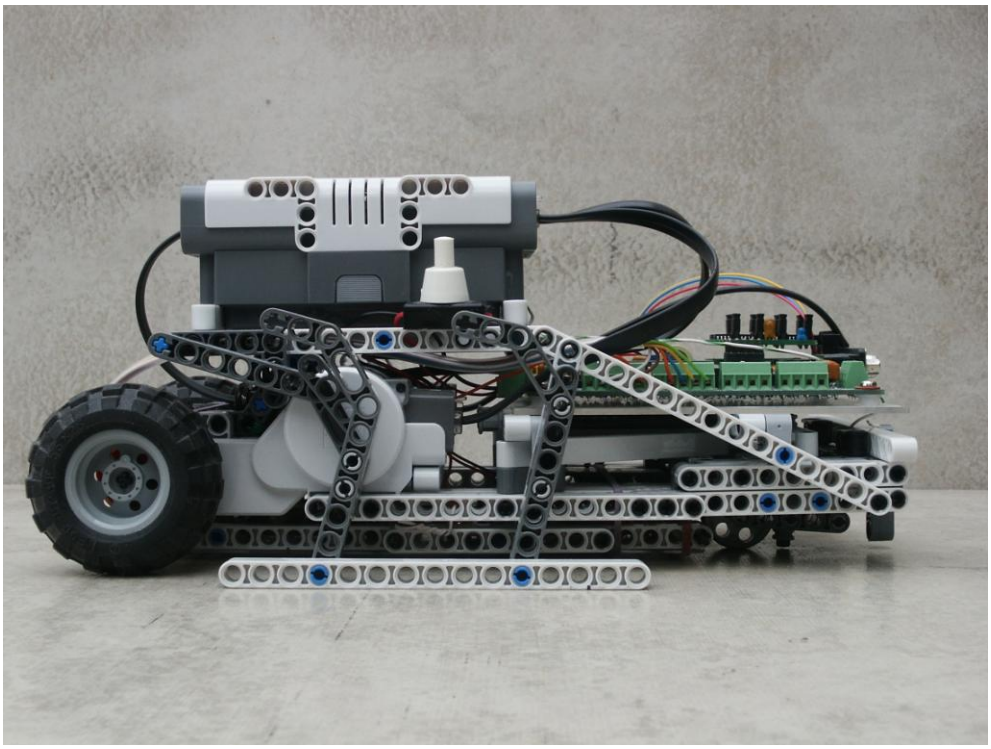


Fig. 8.22 Side view of fully-assembled vehicle

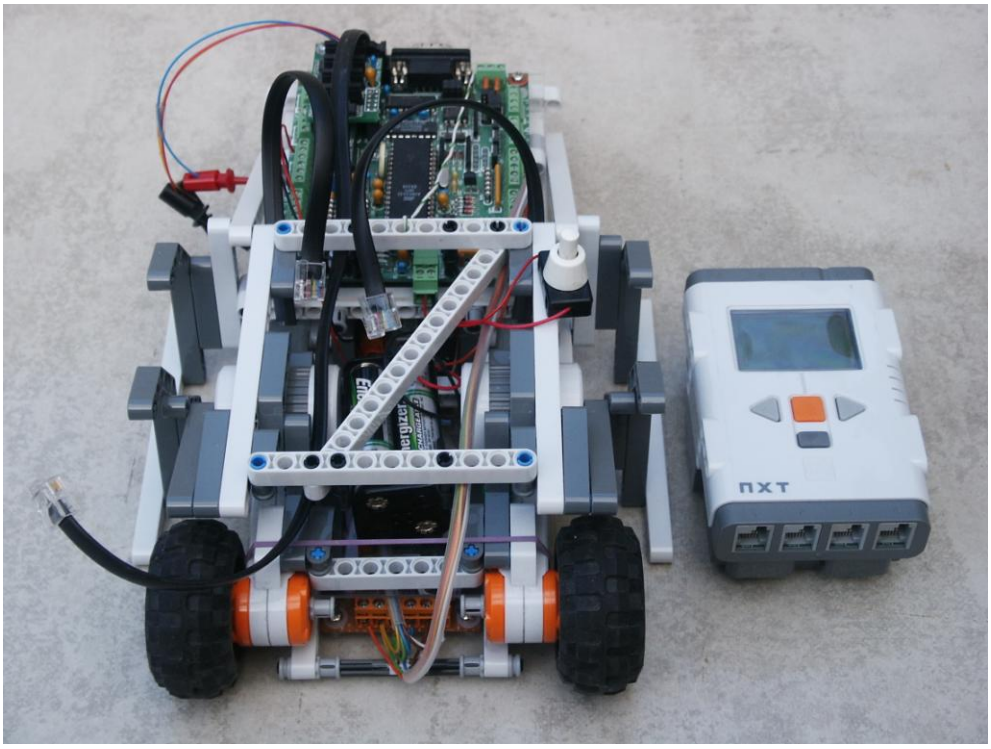


Fig. 8.23 NXT brick removed to show battery compartment

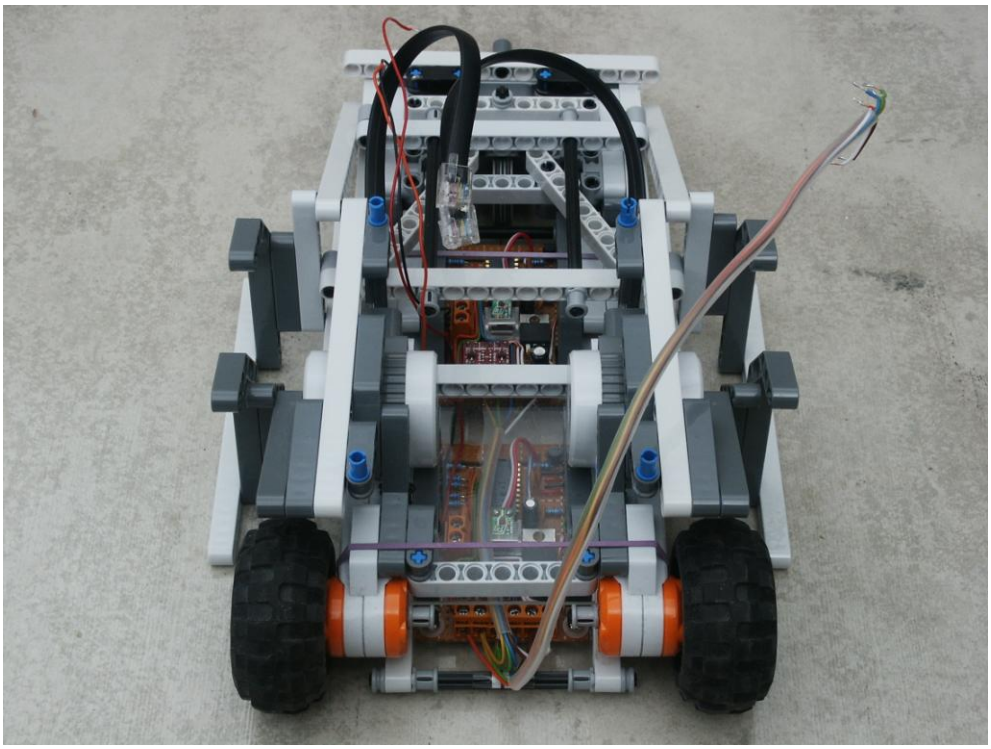


Fig. 8.24 Sensor subsystem exposed

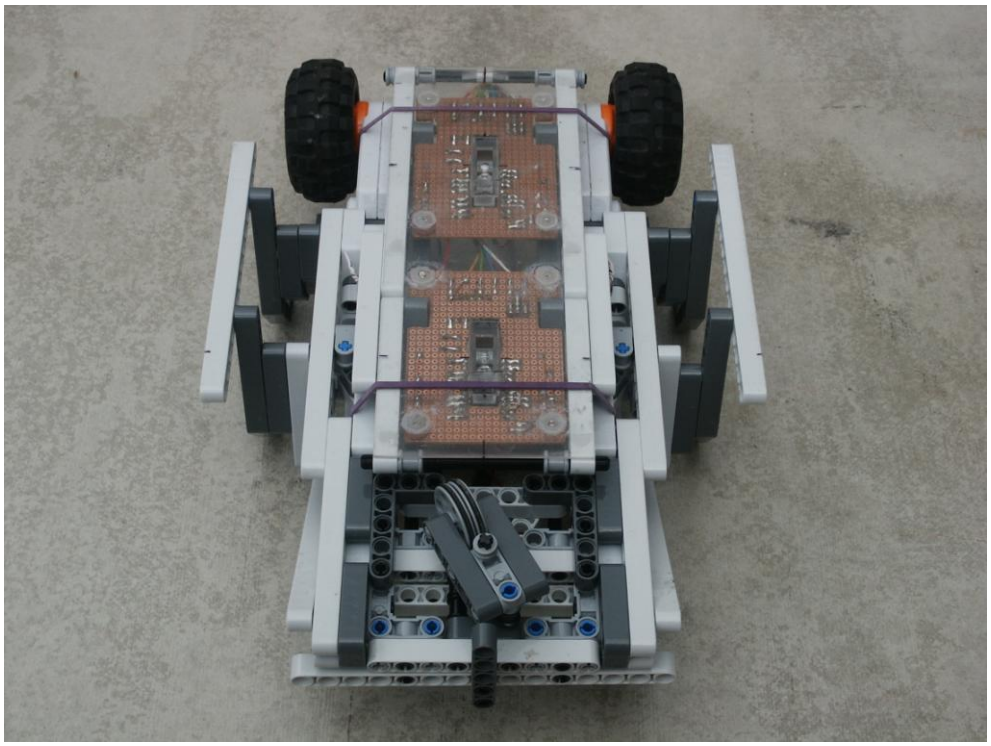


Fig. 8.25 Underside of vehicle

8.2 Software Platform

The main programming environments were MATLAB and Simulink with the inclusion of Real-Time Workshop Embedded Coder. The PID model was developed using MATLAB 2007a while the adaptive model was formulated using version 2010b. The Simulink blocks for interfacing with the Lego Mindstorms NXT are included in an open-source add-on module called the Villanova University Lego Real Time Target (VU-LRT) library blockset as shown in Fig. 8.26. It provides a way for the program to interact with the motors and sensors. The optical sensor block in the program was actually a customised block with adapted code borrowed from the I^2C block of the VU-LRT blockset. The VU-LRT blockset is derived from the Embedded Coder Robot NXT library blockset originally developed by Takashi Chikamasa. Part of the Embedded Coder Robot NXT library blockset is included in Simulink since version 2012a.

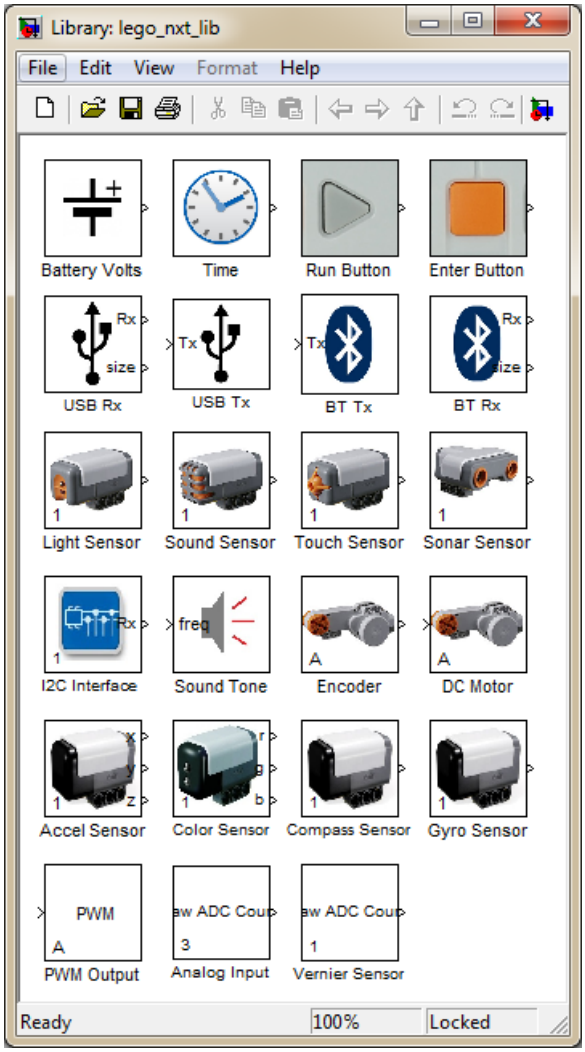


Fig. 8.26 VU-LRT library blockset for Simulink

For the NXT brick, the original firmware was initially replaced by a modified version called the Enhanced NXT firmware. It is developed and maintained by John Hansen. This firmware adds features to the original and also allows both standard and non-standard Lego programs to run on the NXT. In order for Simulink programs to run on the NXT, they must be compiled for the nxtOSEK platform. As the program size became quite large in latter stages of the research, the Enhanced NXT firmware was replaced by the nxtOSEK NXT BIOS. This mode allowed the usage of flash memory and SRAM to be maximised. This meant that larger programs were allowed, and more variables could be stored in the RAM. However, only one program could be stored in the flash memory at any time.

The nxtOSEK platform is an open-source real-time operating system (RTOS) that consists of leJOS device drivers, TOPPERS/ATK (formerly known as TOPPERS/OSEK), and TOPPERS/JSP. leJOS is an alternative open-source firmware that includes a Java virtual machine which allows the NXT to be programmed in the Java programming language.

TOPPERS/ATK is an open-source kernel with real-time multi-tasking capabilities that is used in the automotive industry. TOPPERS/JSP is a real-time kernel that conforms to µITRON4.0 (Japanese industry standard) specifications. nxtOSEK is developed and maintained by Takashi Chikamasa.

In order for Simulink to compile a program for the nxtOSEK platform, it must utilise additional tools such as the GNU ARM toolchain which includes GCC (GNU C Compiler), and Cygwin, which provides a Unix-like environment within the Windows operating system. Both tools are open source.

The ATmega32 microcontroller on the AVR200 board was programmed using the Atmel AVR Studio 4 integrated development environment. This is a proprietary program available from Atmel at no cost. The required additional tools are WinAVR, which includes AVR-GCC (GNU C Compiler for AVR core), the AVR-Libc library, and the Procyon AVRlib C function library. All the ancillary tools are open source.

8.3 Summary

A wheeled robot prototype has been successfully built by using the Lego Mindstorms NXT system. The NXT platform provided the construction pieces, a "brick" containing the CPU and sensor interfaces, and a pair of servo motors for driving the wheels. Additional components include two Avago optical mouse sensors, and an AVR200 processor board equipped with an Atmel ATmega32 microcontroller that provided the necessary interface for communications between the NXT's CPU and mouse sensors. Although all the components are stock items, several modifications had to be made in order to get them to operate as a unified system. As for the software, while MATLAB and Simulink were collectively the primary programming environment, a number of other open source and free proprietary software were required to work in conjunction with the main program in order for the whole system to function.

While the prototype was being assembled, simulation tests were carried out concurrently. Now that the construction of the wheeled robot is complete, hardware validation will be the next logical step. The results of the both simulation and real-world tests will be presented in the following chapter.

Chapter 9: Simulation and Validation Results of the Adaptive Model

The tracking accuracy of a WMR does not only depend only on the precision its sensor. Without a competent control system, the WMR would be rather useless. So, the computational model developed in Chapter 7 will have to undergo extensive testing in order to assess its suitability for the intended purpose. The first test phase will be carried out completely in simulation. Various trajectories will be assigned to the program, and the resulting simulated response will be used to tweak the model to improve its performance. The second test phase will involve real-world testing of the actual prototype. The final results will be presented and analysed in this chapter.

9.1 Simulation Results and Analysis

The trajectories in this chapter are selected to match the ones used in the PID model in order for comparisons to be made. However, there are a few additional simulations here that do not have matching counterparts in the PID model. They are included in order to provide a more comprehensive testing regime. In a couple of the extra tests, the PID model could not perform the simulations to an acceptable standard, so they were not included in the report. It is recommended that Chapter 5 be revisited for comparison purposes.

Looking at the trajectory plot in Fig. 9.1, it is clear that the controller successfully managed to follow the designated path. This is confirmed by the results error distribution results shown in Fig 9.2. The resulting RMS errors are 25.97 mm in the X-axis and 31.28 mm in the Y-axis. This is quite acceptable, considering that the WMR was frequently lagging. The reason for this is that the speed of the prescribed trajectory was set to be near the maximum level allowed for that the WMR. So, whenever the WMR had to slow down when negotiating corners, it faced a mighty task in catching up afterwards. Despite this, it managed to complete its test without any issue.

Another thing is that the controller was forbidden to engage in reverse motion for any of the wheels (for reasons described previously). This caused the turning radius of the WMR to be increased and had a noticeable effect when going around sharp corners. This is the reason that the WMR was seen to veer off temporarily from the expected path while negotiating 90-degree turns. Needless to say, this had an impact on the calculation the RMS errors.

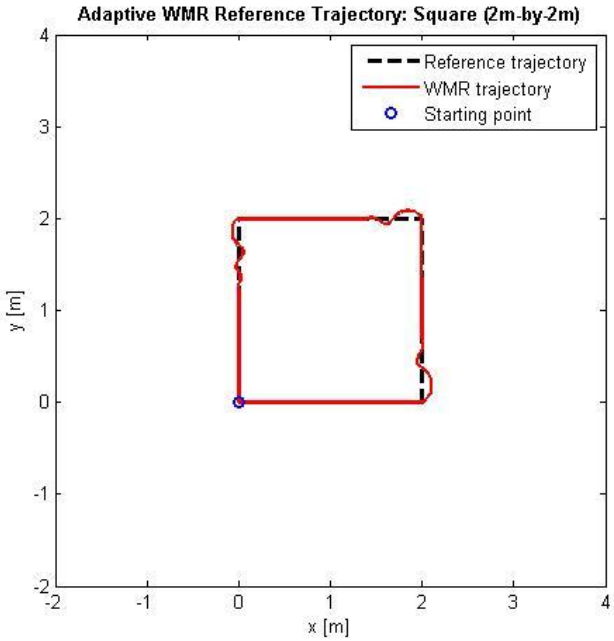


Fig. 9.1 Simulation Test No. 1: Square trajectory (2m x 2m)

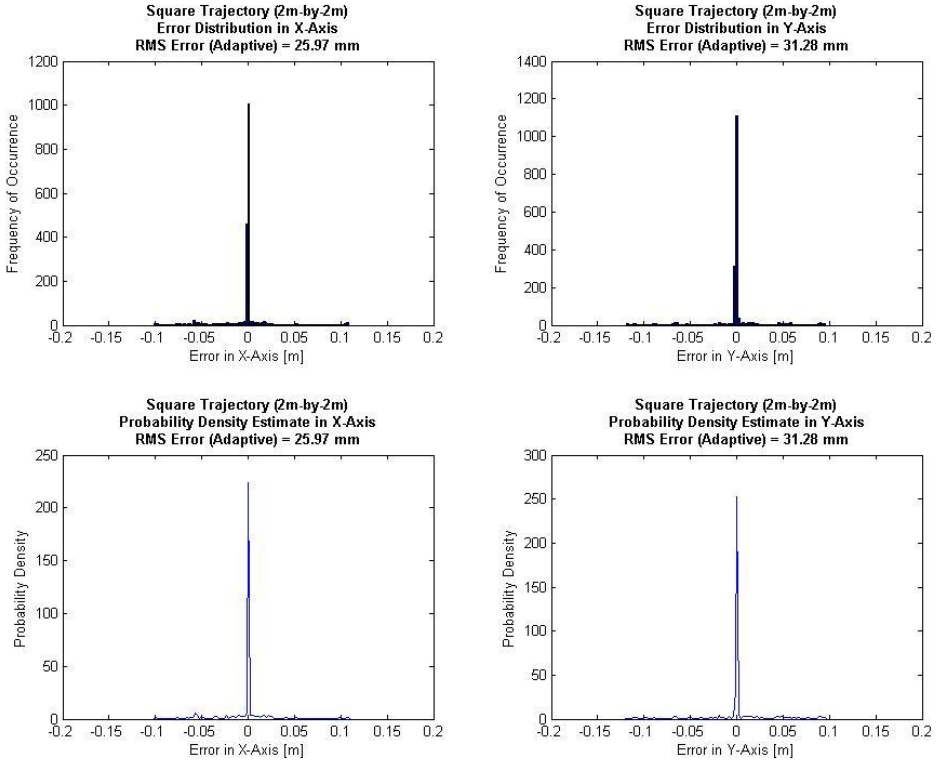


Fig. 9.2 Adaptive Simulation Test No. 1: Square trajectory (2m x 2m) error analysis

The plot shown in Fig. 9.3 is an enlarged version of the previous test. It is also the same path as used in the UMBmark calibration test as mentioned in the chapter on PID simulation results. The outcome shows that that the controller was able to follow the designated path very well. The error distribution results are shown in Fig 9.4. The resulting RMS errors are 17.63 mm in the X-axis and 19.05 mm in the Y-axis. The longer stretches of straight lines compared to the preceding test meant that the WMR had more time to pick up speed after being slowed down by sharp corners. These error values are clearly superior to those in the PID test. Also, only a slight tweak to one of the gain settings was necessary to obtain these results. Indeed, the controller performed quite well even without the adjustment. So, the tweak was more like a bit of fine-tuning.

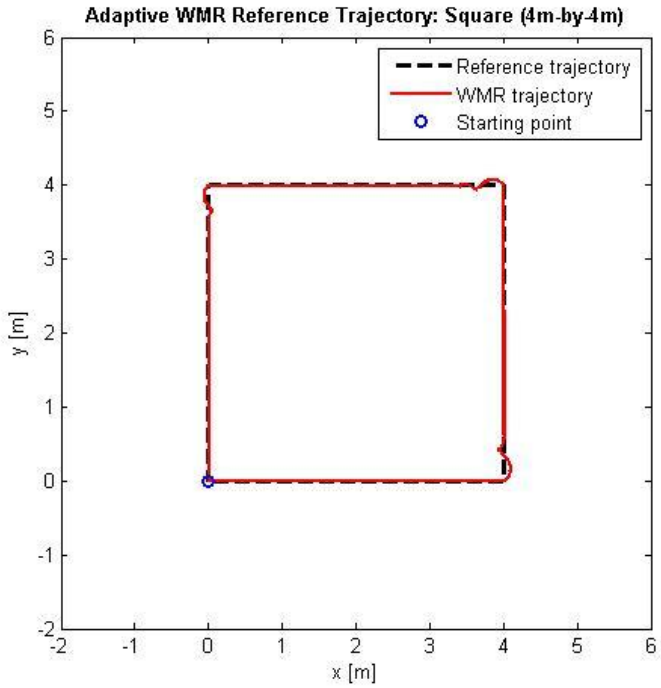


Fig. 9.3 Adaptive Simulation Test No. 2: Square trajectory (4m x 4m)

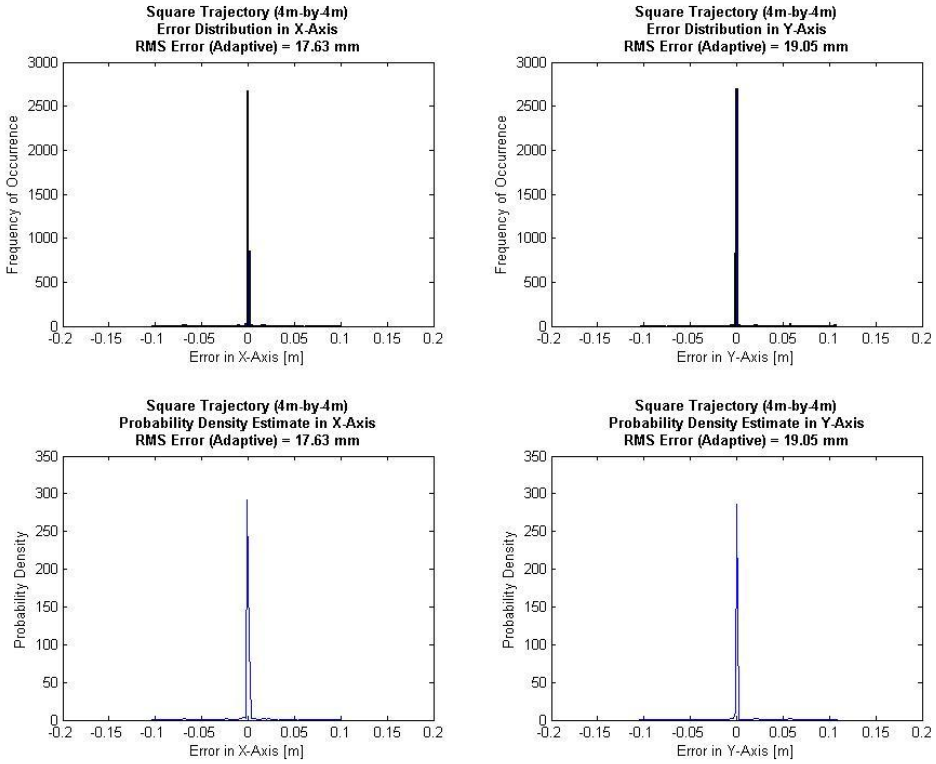


Fig. 9.4 Adaptive Simulation Test No. 2: Square trajectory (4m x 4m) error analysis

The plots shown in Figs. 9.5 and 9.6 share the same course as the preceding test. However, the starting positions of the WMR and trajectory are different. This meant that the WMR lagged the trajectory from the beginning and had to play catch-up. This was a good challenge for the controller just like it was in the similar PID test. The end result is excellent as demonstrated by the WMR's ability to steer toward the path and maintain its trajectory afterwards. In contrast to the PID test, there was no need at all to re-tune the controller's gain settings from the previous test. The versatility of the adaptive controller is clearly evident. As in the similar test in the PID simulation, there was no error analysis conducted for these scenarios because the offsets would skew the results. Besides, the ability of the controller to track this particular trajectory had already been tested in the preceding simulation.

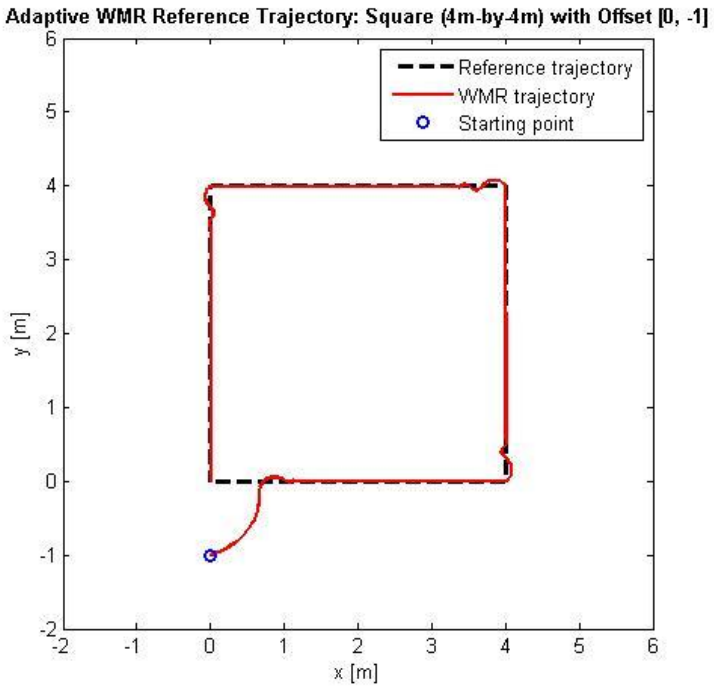


Fig. 9.5 Adaptive Simulation Test No. 3: Square trajectory 1 with offset starting point

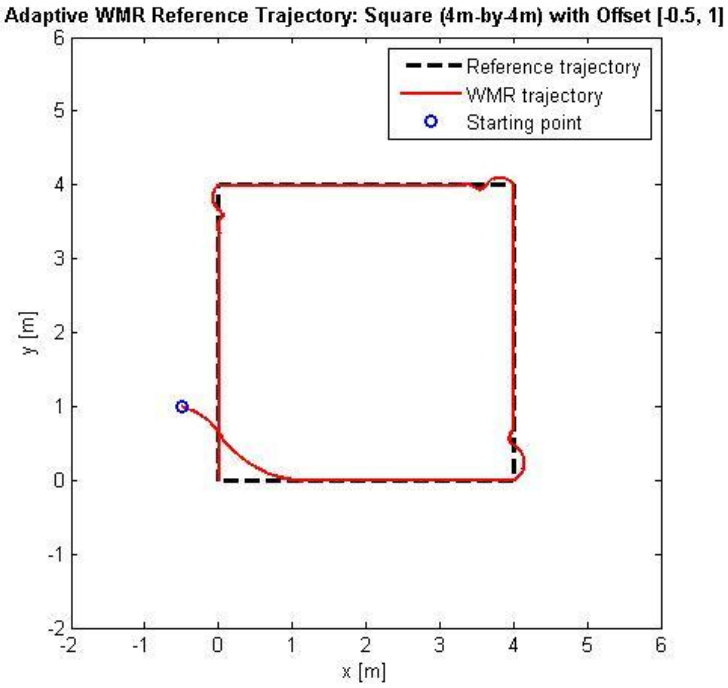


Fig. 9.6 Adaptive Simulation Test No. 4: Square trajectory 2 with offset starting point

The sinusoidal trajectory lacked the sharp turns that were present in the square paths. However, straight lines are easier to track than curves. In the end, the gentle curvature of the sinusoid did not turn out to be much of a challenge to the controller's ability at all. As is evident from the plot in Fig. 9.7, the WMR tracked its assigned trajectory with ease. This observation was backed up by the error analysis charts in Fig. 9.8. The RMS errors in the X- and Y-axes are 2.63 mm and 5.60 mm respectively. These figures are simply outstanding and far surpassed the results in the equivalent PID test. To obtain the best results, a slight adjustment was made to one of the gain settings from the previous 4m-square test. Indeed, the final settings ended up being identical to the ones used in the 2m-square simulation - which were only marginally different from those used in 4m-square test anyway. Yet again, even under varying situations, the controller continued to demonstrate its remarkable adaptive qualities.

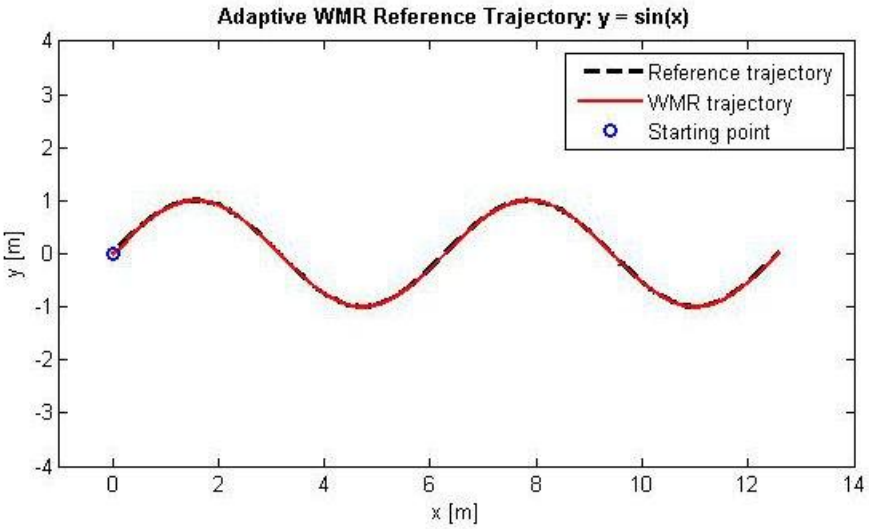


Fig. 9.7 Adaptive Simulation Test No. 5: Sinusoidal trajectory

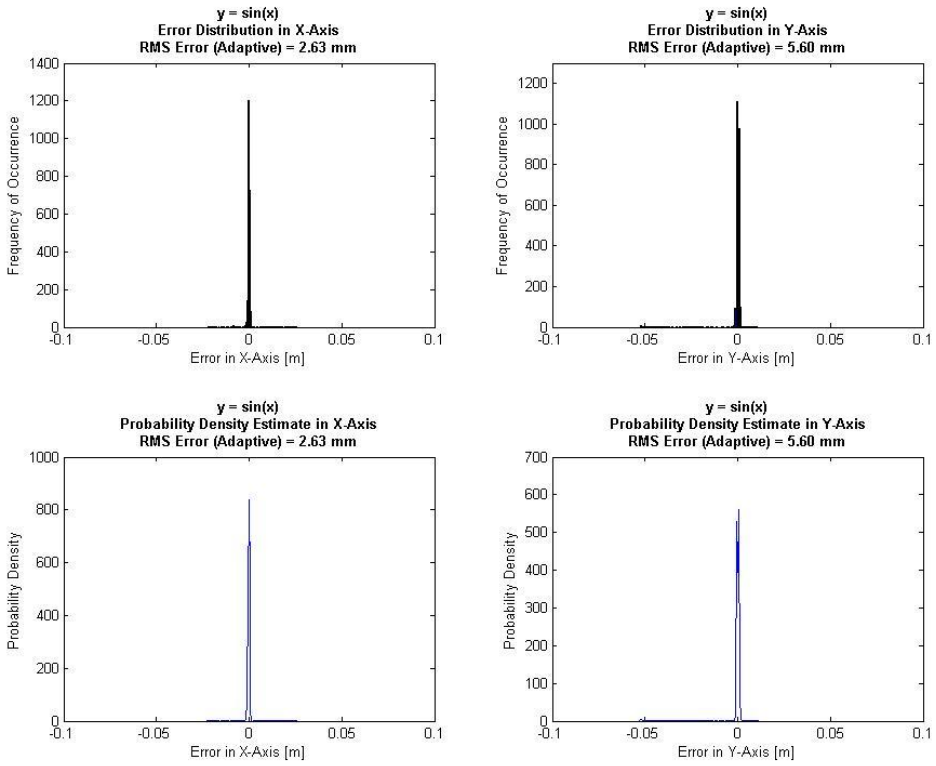


Fig. 9.8 Adaptive Simulation Test No. 5: Sinusoidal trajectory error analysis

The trajectory from the previous test was run again using a sinusoid of double the magnitude and frequency as shown in Fig. 9.9. Just as in the similar PID test, the length of this course was slightly over twice that of the preceding sinusoidal path, and the assigned completion time was doubled. This would prove to be quite a challenge for the controller because the more acute turns will slow the WMR quite dramatically, but no additional time was given for it to recover its speed after negotiating those turns. Despite the demanding circumstances, the controller was able to complete the task without any problem as is illustrated by the trajectory plot.

Looking at the error distribution and density charts in Fig 9.10, it is clear that the results were substantially worse than the previous sinusoidal test. Given the more arduous task, the outcome is to be expected. The RMS errors in the X- and Y-axes are 21.75 mm and 60.92 mm respectively. The performance is still unmistakeably superior to the PID model, although the difference is not as substantial as in other tests. This probably indicates that even though the adaptive controller performed better than the PID model, it still found the task quite challenging.

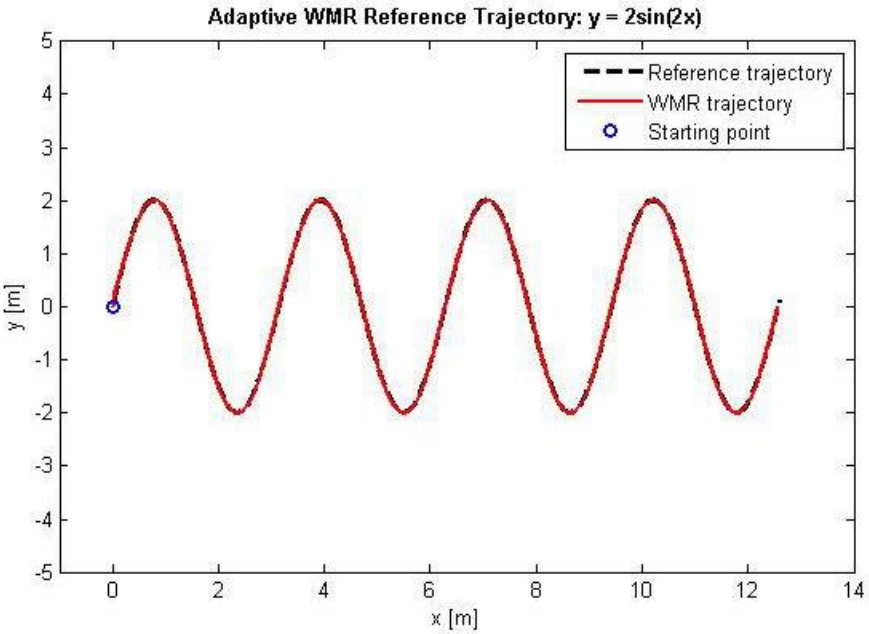


Fig. 9.9 Adaptive Simulation Test No. 6: Sinusoidal trajectory 2

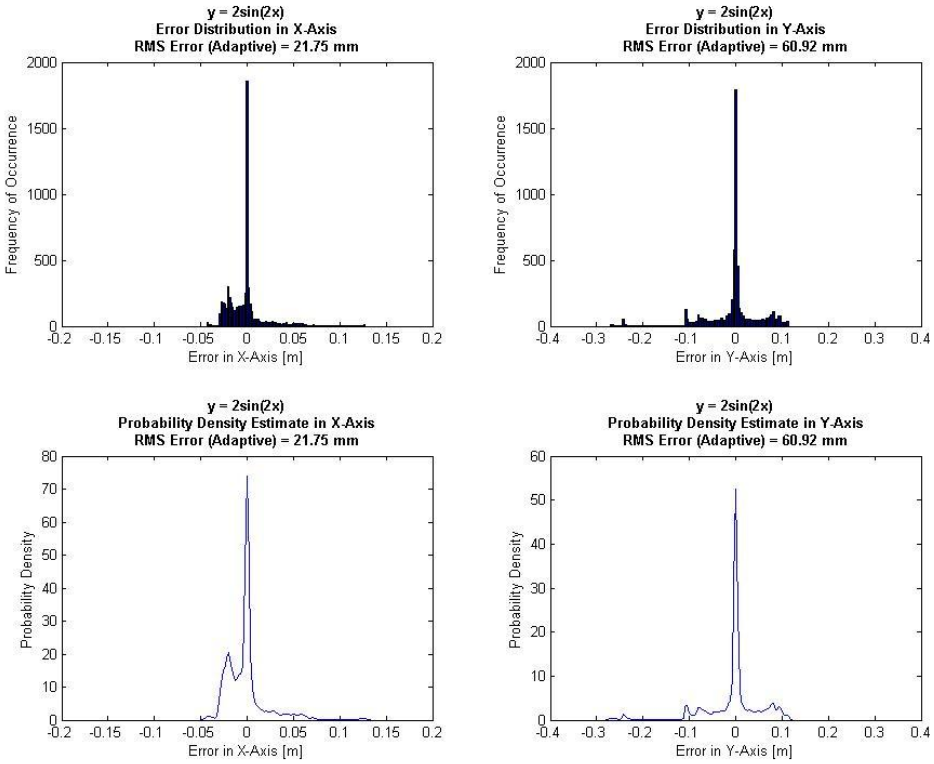


Fig. 9.10 Adaptive Simulation Test No. 6: Sinusoidal trajectory 2 error analysis

The triangular trajectory show in Fig 9.11 was probably one of the more challenging tests for the WMR's controller. The sharp and acute angles of the turns made the negotiation around them rather tricky and slow. It was not helped by the fact that the WMR was not allowed to reverse its wheels, so the turning radius was relative large. All this meant that the WMR often lagged behind the trajectory. In spite of all these factors, the controller still managed to get the WMR to stick to the trajectory quite well as evidenced by the trajectory plot.

Due to the wide turns taken by the WMR, quite a bit of path deviation was expected in the final outcome. Indeed, the results of the error analysis confirmed the prediction. The RMS errors in the X- and Y-axes are 56.39 mm and 60.90 mm respectively as shown in Fig. 9.12. Compared to the results from previous simulations with less demanding trajectories, these numbers are not too bad at all. The straight lines probably helped the situation because they are easier to follow and also allow the WMR to pick up speed much quicker.

This trajectory was the first to require a tweaking of more than one gain setting in order to get the best results. However, the adjustment process took only a short time to complete as opposed to the typical PID tuning routine. Indeed, this test was also carried out for PID model, but no amount of tuning could get the controller to perform adequately. So, the results were left out.

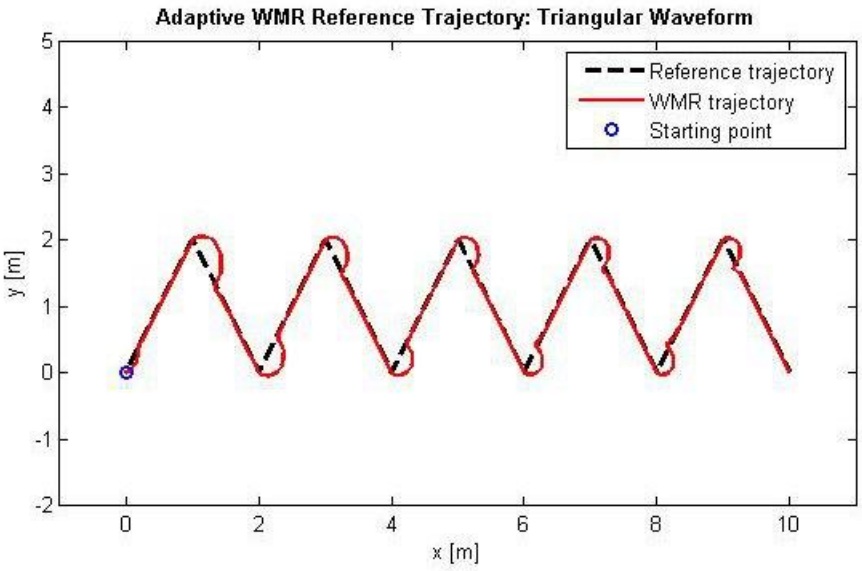


Fig. 9.11 Adaptive Simulation Test No. 7: Triangular trajectory

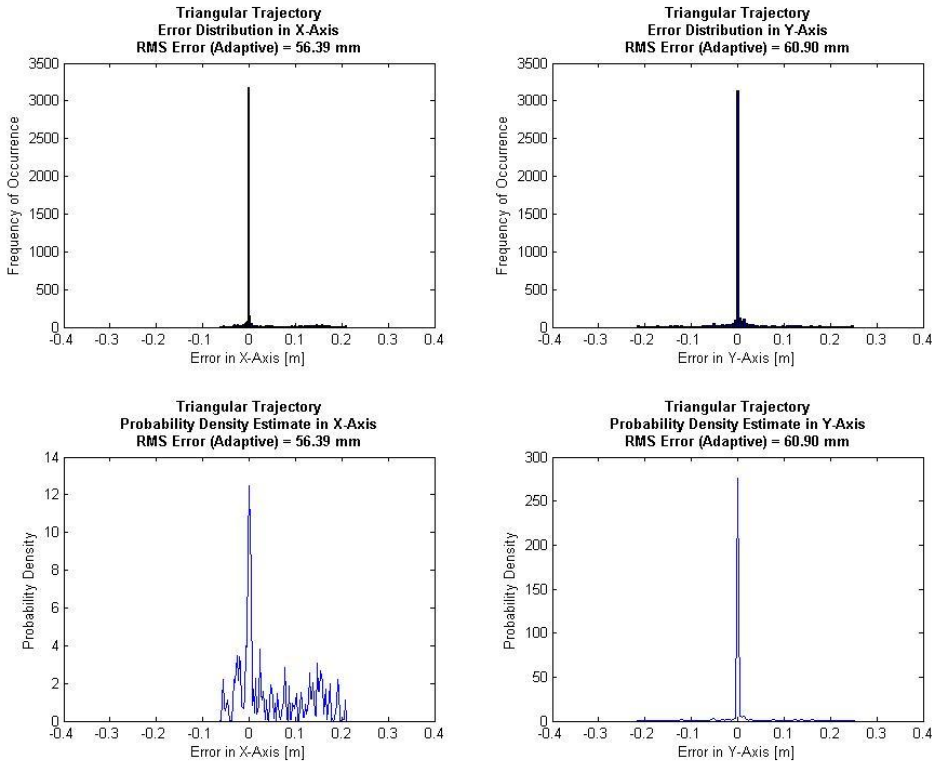


Fig. 9.12 Adaptive Simulation Test No. 7: Triangular trajectory error analysis

The circular trajectory as shown in Fig 9.13 shares one obvious characteristic with the square path, which is they each start and end at the same place. The deviation from the original point provides a good visual indication of much tracking drift or error has been accumulated. Although there are no sharp corners to negotiate, tests have shown that straight lines are much easier to track and navigate than curves.

In order for direct comparisons with the PID model, the test was conducted for both one and two revolutions of the circle. Unlike the PID test, there was no instance where the WMR was able to perform adequately for one revolution but suffered from oscillations as the test continued. In order to prevent an overlapping plot from obscuring any undulation or minor detail of the WMR's trajectory, the plot for one revolution will also be shown here.

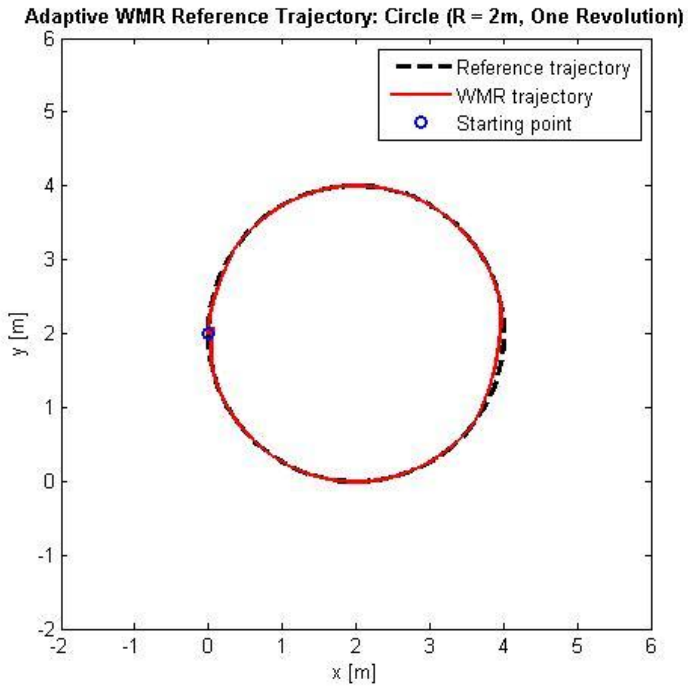


Fig. 9.13 Adaptive Simulation Test No. 8: Circular trajectory (r = 2m, 1 rev)

The plot in Fig. 9.14 shows that the controller was fully capable of maintaining a circular trajectory for two revolutions. Some imperfections in the WMR's trajectory were noticeable but not very significant. A clearer picture of the controller's performance is provided by the error analysis in Fig. 9.15. The RMS error of 19.48 mm in the X-axis is quite comparable to those of previous simulations with different trajectories. Whereas, the RMS error of 119.19 mm in the Y-axis is far poorer than any of the adaptive simulation results collected thus far. However, since the error spread is quite small and the trajectory plot indicates very little path deviation, most of the error could be attributed to lag. This is somewhat expected because a curved trajectory is harder to track than one with straight lines. But the amount of lag is a bit more than anticipated. However, considering that the total distance travelled in the Y-axis was 4 m, the error margin is still quite acceptable. The performance is clearly superior to the PID-based model, but not quite as much as in other tests.

Another point is that the prescribed trajectory speed was almost 90% of the WMR's maximum velocity. That meant that if there were any minor tracking difficulties, the WMR would struggle to keep up with the trajectory. This might be a possible explanation for the unexceptional results. Despite the issues with lag, it is important to note that the controller was able to keep the WMR on track throughout the test without much deviation.

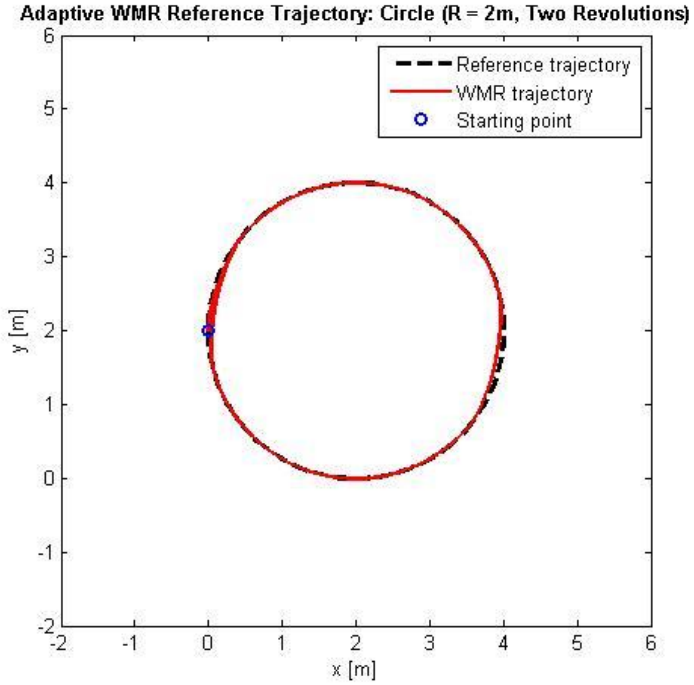


Fig. 9.14 Adaptive Simulation Test No. 8: Circular trajectory (r = 2m, 2 revs)

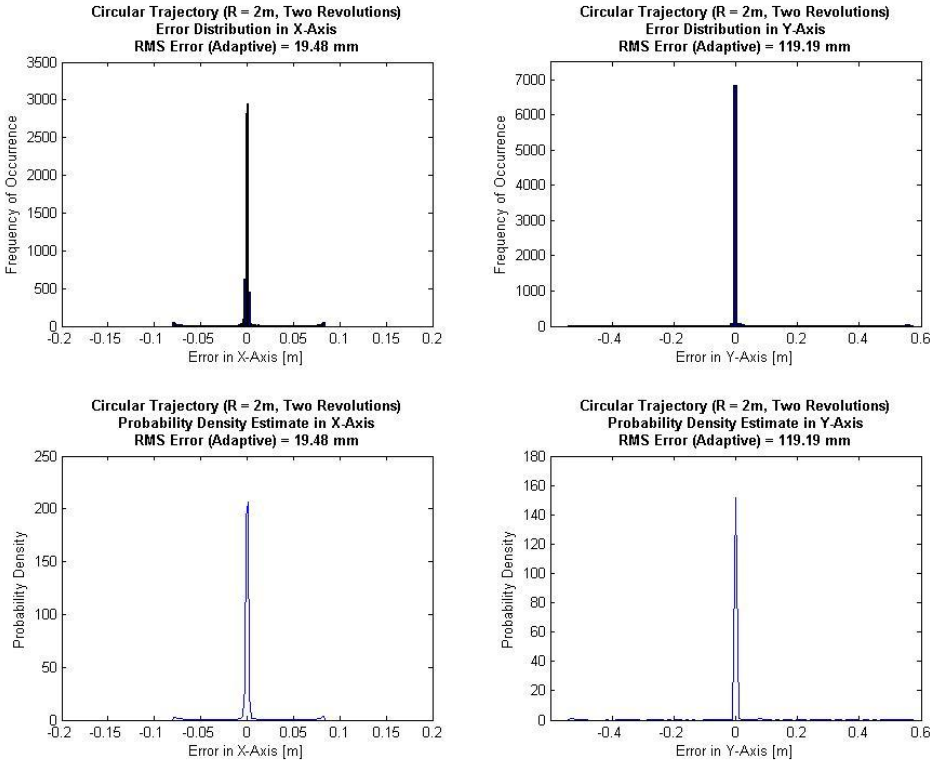


Fig. 9.15 Adaptive Simulation Test No. 8: Circular trajectory (r = 2m, 2 revs) error analysis

The simulation shown in Fig. 9.16 is basically the same as the preceding test except that the trajectory is a larger circle. Just as before, the visual results indicate that the controller performed well in maintaining its assigned trajectory. The error analysis charts in Fig. 9.17 will provide a better gauge of the actual performance of the controller. Compared to the previous test, the results here were even less stellar. An RMS error of 33.14 mm in the X-axis is still quite acceptable. However, a discrepancy of 199.34 mm in the Y-axis is quite poor. Since there was little deviation of the WMR from the assigned trajectory, the errors must be due to lag once again.

The error distribution shows that the WMR managed to keep up with the trajectory for the most part, but there are occasions when it lagged by quite a significant margin. It is postulated that if the prescribed trajectory speed is lowered, the lag issues would be reduced or eliminated. In any case, this is a good way to test the limits of a controller's abilities. Since the WMR was able to follow its preset trajectory without veering off, the controller must be considered to have done its job - albeit not quite perfectly.

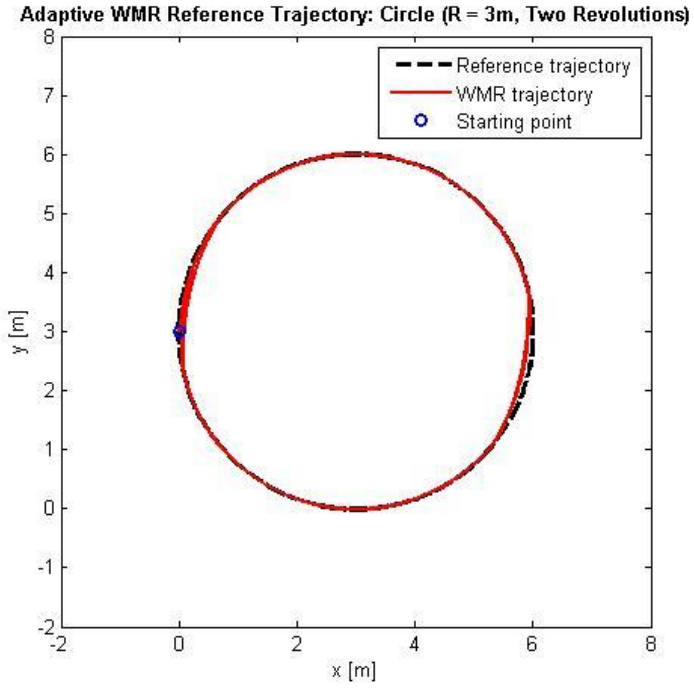


Fig. 9.16 Adaptive Simulation Test No. 9: Circular trajectory (r = 3m, 2 revs)

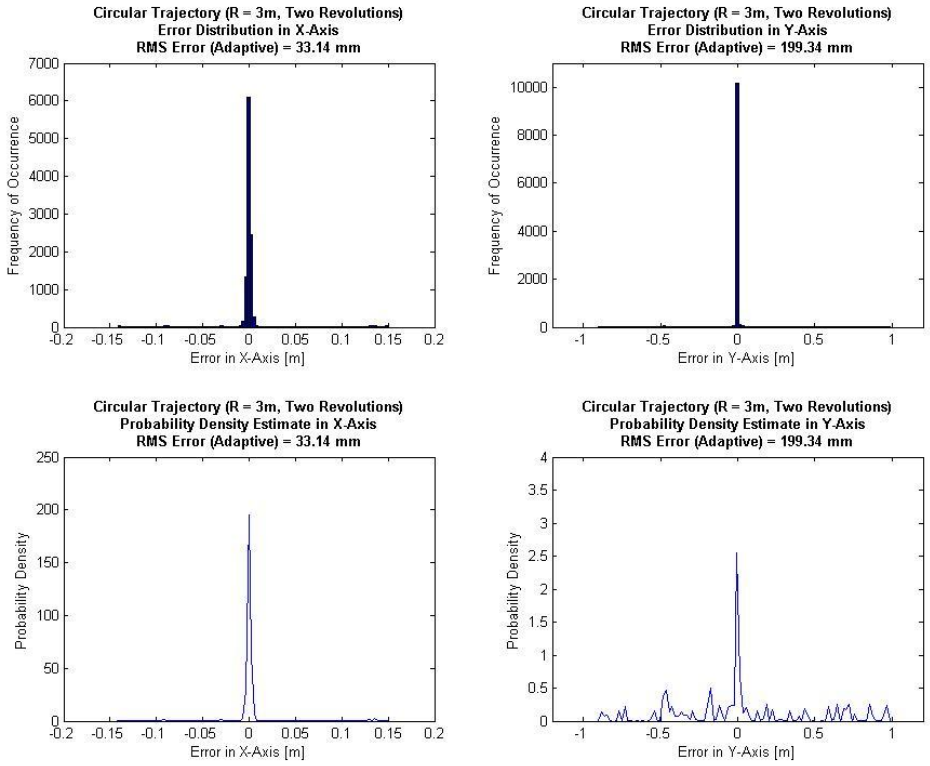


Fig. 9.17 Adaptive Simulation Test No. 9: Circular trajectory (r = 3m, 2 revs) error analysis

At the conclusion of the battery of simulation tests, a summary of the results is presented in Table 9.1 for ease of reference.

Trajectory Type	RMS Error in X-Axis (mm)	RMS Error in Y-Axis (mm)
Square: 2m x 2m	25.97	31.28
Square: 4m x 4m	17.63	19.05
Sinusoidal: y = sin(x)	2.63	5.60
Sinusoidal: y = 2sin(2x)	21.75	60.92
Triangular	56.39	60.90
Circular: radius = 2m	19.48	119.19
Circular: radius = 3m	33.14	199.34

Table 9.1 Summary of simulation test results of adaptive model

9.2 Hardware Validation Results and Analysis

When the algorithm of a tracking controller has been thoroughly tested in simulation, the logical steps that follow would be to build a prototype to verify the computed predictions. The vehicular prototype is based on the Lego Mindstorms NXT platform. Details of the vehicle and its main components are described in Chapter 8.

The WMR used here has only two optical mouse sensors. It has no access to additional on-board or external sensors that may provide any kinds of positional or inertial information. Due to this restriction, it is impossible to track the actual movement of the vehicle. However, there is no problem in physically measuring its final resting coordinates because it position can be marked on the floor. Indeed, this is the most important point. During operation, a WMR generally has some allowance for minor deviation from its assigned trajectory. However, the main objective is for it to get as close as possible to its destination coordinates. If the WMR is able to consistently reach its assigned destination coordinates accurately, it means that it is able to track its position very well. This infers that the WMR is also able to follow its prescribed trajectory with good precision. It is highly unlikely that a WMR could repeatedly arrive at the final coordinates but not be able accurately track any other point along its trajectory.

Noting the lessons learnt from the simulation tests, the assigned trajectories used in the actual tests would only be about 70% of the maximum speed of the WMR. This will allow it the opportunity to catch up after it has performed a manoeuvre that has reduced its speed. Otherwise, it may not be able to reduce or eliminate the lag caused by a slowdown, and these lags could accumulate if the prescribed trajectory is quite convoluted. Since the WMR did not encounter any scenario where making a trajectory correction would risk the vehicle running backwards for prolonged periods of time, the restrictions on reverse motion was lifted in order to allow tighter turning manoeuvres.

The initial set of tests is a basic evaluation of the tracking performance of the WMR over straight line of 2 m, i.e. $x = 2$ m and $y = 0$ m. It also includes a comparison of the two localisation methods used, i.e. kinematic and hybrid geometric.

Judging from the outcome of the first round of tests shown in Table 9.2, the discrepancy between the two approaches were quite small considering the distance travelled. However, the differences were not totally insignificant. The hybrid method displayed a smaller mean error and RMS deviation than the purely kinematic technique. So, the hybrid method is clearly a better - at least for this particular test.

	Dual-Sensor Tracking (Trajectory: x = 2 m; y = 0 m)			
	Hybrid Geometric Localisation		Kinematic Localisation	
	Error in X [mm]	Error in Y [mm]	Error in X [mm]	Error in Y [mm]
1	-5	2	-9	4
2	4	3	8	7
3	-2	3	2	6
4	4	0	5	-2
5	4	2	9	7
6	2	-4	2	-2
7	-3	-4	-2	-4
8	-3	-2	-3	6
9	-4	-2	-7	4
10	2	0	2	-2
Mean error [mm]	-0.100	-0.200	0.700	2.400
RMS error [mm]	3.450	2.569	5.701	4.796

Table 9.2 Tracking errors in dual-sensor configuration

The objective of the next set of tests is to evaluate the tracking performance if only one sensor is utilised. The same trajectory would be used for comparison with the dual-sensor system. In this scenario, the WMR can only use the kinematic method for localisation purposes.

The results of the single-sensor test are displayed in Table 9.3. It is clear that the performance of the WMR when adopting this configuration was noticeably poorer than when employing the dual-sensor system no matter which localisation method was used in the latter. The mean error in the direction of travel is very small, but there is quite a bit of variance in data. As for the lateral direction, the mean error is significantly greater than in the longitudinal direction and the variance is even larger. The deviation of WMR became slightly noticeable towards to end of the trajectory. This was most likely due to the errors that had been accumulated.

The inferior results compared to the previous tests are not entirely unexpected. After all, this is the rationale for using two sensors instead of one, and the prediction has been confirmed. It should be kept in mind this is the method used as a contingency when data from one of the sensor is deemed unreliable. As discussed in earlier chapters, this approach should not be

relied upon for localisation purposes, and any usage should be infrequent and only for very short periods of time.

	Single-Sensor Tracking (Trajectory: x = 2 m; y = 0 m)	
	Error in X [mm]	**Error in Y [mm]**
1	-25	10
2	16	31
3	2	29
4	9	7
5	13	39
6	11	5
7	-5	-2
8	-7	26
9	-20	13
10	-2	9
Mean error [mm]	-0.800	16.700
RMS error [mm]	13.168	21.040

Table 9.3 Tracking errors in single-sensor configuration

The third set of tests puts the WMR through a square trajectory with dimensions of 4 m on all sides. As mentioned previously, this is the same course used in the UMBmark calibration test. It must be noted that this WMR uses an optical sensor system that is calibrated very differently from those using wheel encoders. However, it is not the calibration procedure that is of interest here. The UMBmark test has been used quite frequently by other researchers, so their final results can be used as a comparison to the ones obtained in this test. This test would be run five times in the clockwise direction and five times in the anticlockwise direction.

The results shown in Tables 9.4 and 9.5 indicate that the tracking and localisation capabilities of the WMR are very good. The largest mean error in any direction is only 13.2 mm. However, its error spread is quite a bit bigger, with a maximum of at 33.711 mm. In any case, this is still an excellent result.

	Clockwise Direction in Square Trajectory (4m x 4m)	
	Error in X [mm]	**Error in Y [mm]**
1	37	35
2	-27	-25
3	-18	34
4	31	28
5	39	-17
Mean error [mm]	12.400	11.000
RMS error [mm]	31.318	28.562
RMS error [%]	0.391	0.357

Table 9.4 Tracking errors in square trajectory (clockwise)

	Anticlockwise Direction in Square Trajectory (4m x 4m)	
	Error in X [mm]	**Error in Y [mm]**
1	21	-29
2	30	36
3	-29	43
4	-35	36
5	-38	-20
Mean error [mm]	-10.200	13.200
RMS error [mm]	31.148	33.711
RMS error [%]	0.389	0.421

Table 9.5 Tracking errors in square trajectory (anticlockwise)

The method used by UMBmark for error quantification is the magnitude of the offset from the mean error. The larger of the clockwise and anticlockwise results would be the final defining value.

$$r = \sqrt{\bar{x}_{err}{}^2 + \bar{y}_{err}{}^2} \tag{9.1}$$

where $\bar{x}_{err}$ is the mean error between computed and actual positions in x

$\bar{y}_{err}$ is the mean error between computed and actual positions in y

Using the metric describe above, the error magnitude for both the clockwise and anticlockwise tests are:

$$r_{cw} = 16.58\ mm \tag{9.2}$$

$$r_{acw} = 16.68\ mm \tag{9.3}$$

$$E_{\max} = \max\left(R_{cw}\ ;\ R_{acw}\right) \tag{9.4}$$

$$E_{\max} = 16.68\ mm \tag{9.5}$$

The outcome according to this metric is very good in comparison to other dead-reckoning systems. The result achieved by another dual-sensor system for the same trajectory is an error of 114 mm (Bonarini et al. 2004). The best results attained by the developer of the UMBmark over eight sets of tests are errors that ranged from an average of 12 mm to 35 mm with a standard error of the mean of 11.2 mm (Borenstein and Feng 1996). However, the WMR was painstakingly calibrated (and sometimes had to be re-calibrated between sets) and could only run at a slow pace in order to avoid tyre slippage. That is because it relied on wheel encoders for odometric measurements and thus could not compensate for any slippage.

For reference purposes, the performance results of another WMR that used multiple mouse sensors will be mentioned here (Sekimori and Miyazaki 2007). The trajectory used in that case was 2.5 m long and included four right-angle turns. Even though the use of a different trajectory means that the results are not exactly comparable to the ones adopted in this research, it is nonetheless useful to cite them here for some sort of comparison. Using a two-sensor configuration, it produced an average translation error of 44.516 mm. Using a four-sensor configuration, it produced an average translation error of 38.011 mm.

The sensor calibration process used by the WMR in this research was simple and straightforward. The procedure only needed to be performed once for a particular type of tracking surface. Without needing further calibration or adjustment, the WMR was able to repeat its good performance over and over again. The effectiveness of the tracking and localisation system used by the WMR in this research was thus confirmed.

The last round of tests is an assessment of how well the redundant system works. The idea is to disrupt the tracking of either of the sensors (but not both) for a very short period of time. The sensor cannot focus properly if the gap between itself and the tracking surface is too small. A thin object that could slide underneath the sensor mounting plate would be a good choice at first thought. However, since the both sensors are lined up behind the front castor wheel, it means that castor wheel would encounter the object as well. If this happens, the front of the vehicle would be lifted, and the tracking of both sensors would be affected as a

result. The only way to do this is to introduce the item right behind the castor wheel before the WMR begin moving. However, it would be limited to one attempt per test run.

It has been said that optical mouse sensors work poorly on mirror-like surfaces. Since mirrors are not paper thin, aluminium foil was used in a test run instead. Somehow, the sensor had no problem coping with it. The next idea was to slide a ruler under the vehicle quickly and pull out before it makes contact with the rear wheels. It proved to be quite impossible to carry out because the rear sensor is very close to the rear wheels and the ruler kept getting run over by one of wheels before it could be completely pulled out.

Finally, the solution was found in a 15cm-long feather. It was light and flexible, and could be flicked in and out under the vehicle very quickly. There were still plenty of times that it got caught under the rear wheel, but no alternative had been found at the time of experiment. Due to the inexact manner of this method, every test run should be analysed individually.

This set of tests is a straight-line trajectory that runs for 2 m with sensor obstruction affecting front and rear sensors alternately. The obstruction manoeuvre would occur twice during the run and for about one to two seconds each time.

The results in Table 9.6 clearly illustrate the massive amount of errors that can occur if inaccurate data is not properly identified and accounted for. Indeed, when the sensors were obstructed for longer periods of time, the errors were even greater. As stated before, the inexact test conditions means that the individual test runs cannot be collectively analysed. However, on an individual basis, the results are very clear. For the runs where the error compensation algorithm was activated in the main program, the outcome shows that it works quite well. The results are comparable to those from the single-sensor test as shown earlier in Table 9.3.

	Sensors Momentarily Obstructed (Trajectory: x = 2 m; y = 0 m)			
	Without Error Detection		**With Error Compensation**	
	Error in X [mm]	**Error in Y [mm]**	**Error in X [mm]**	**Error in Y [mm]**
1	-95	174	-31	29
2	-81	-125	-25	36
3	-182	146	-9	12
4	-153	113	-19	-23
5	-126	-77	14	8

Table 9.6 Tracking errors with and without error compensation

9.3 Discussion and Conclusion

The simulation results are very good and show that the controller's algorithm was capable of handling the job. It managed to cope with every trajectory that was assigned to it. Even in situations where it could barely keep up because of speed limitations, it still kept tracking the prescribed path with very little deviation. Unlike the PID model, the adaptive system only required very minor adjustments to its gain settings in order to obtain optimum results in various test scenarios. After the controller has proven its abilities, the next step was to build a wheeled robot as a test prototype for verifying the outcomes predicted by the simulations.

The hardware validation results were excellent throughout. For odometry comparison, the hybrid geometric method proved to be superior to that of the Euler-based kinematic approach. For the UMBmark test, the mean errors were quite small and better than expected. However, the margin of RMS error was between two and three times that of the mean error. This reflects the amount of error that is random or unaccounted for. Considering that mean error is small, even with this level of variance the overall performance is still very good compared to other WMRs that are based on a dead-reckoning system. An important caveat regarding the results is that the tracking surface used in the tests was relatively smooth, and the surface texture was quite uniform. The results would certainly be poorer if there had been pronounced undulations in the tracking surface or large variations in the surface texture. It must also be noted that the actual tracking accuracy still cannot match the high level of precision provided by the sensors.

It is suspected that the outriggers had played an inadvertent role in contributing to the stability of the vehicle. They were only meant for marking out the sensor's position on the ground. However, since they were almost touching the ground, they ensured that during vehicle motion a minimum clearance between the sensors and ground was maintained regardless of any minor flexing of the chassis or tyres. The results after the addition of the outriggers seem to indicate a slight but noticeable improvement.

The outcome of the redundancy test was quite satisfactory. It showed that the system as able to detect reading errors and the controller was able to compensate for them. The results were quite comparable to those of the single-sensor test, but perhaps not quite as good. In contrast, the performance of the WMR without any error detection was inferior to the single-sensor test by a wide margin. The disruption of the sensors' tracking was infrequent and only momentary, but it was enough to distort the localisation of the WMR. As dead-reckoning errors are cumulative, the outcome would obviously be worse if the sensors were constantly losing their tracking. However, due to the inherent nature of this method, the results would be much improved if sensor disruptions were only limited to very short periods, e.g. well under 1 s.

Although the WMR has demonstrated its ability to detect and account for sensor error, the contingency algorithm cannot provide the same accuracy offered by a dual-sensor system. So, the level of redundancy must strictly be considered as only partial. However, as the evidence has shown, having partial redundancy is clearly preferable to none at all.

Chapter 10: General Conclusions

At the culmination of this research endeavour, some salient points can be made in regards to the work is presented in this thesis. The research questions that were posed at the beginning of this study will also be addressed in relation to the work that has been conducted.

This research explored two fundamentally different tracking control systems. The initial system employed a PID-based control algorithm which used Euler's approximation for odometric calculations. Two models were designed for this system: one that was purely kinematic and the other that had dynamic constraints imposed. When the dynamic algorithm was activated in the model during simulation, it was clear that the simulated motion slowed down in certain situations. This was not surprising because the purely kinematic model did not have to take into account any dynamic factors that would affect vehicular behaviour, such as acceleration around a bend. Thus, without any dynamic considerations, a purely kinematic model is not quite realistic and is only applicable to slow-moving vehicles. This addresses Research Question 2. In general, the simulation results were mostly acceptable and sometimes quite good. On occasions, they were a little less than satisfactory.

The major drawback of the PID-based system is that the gain settings apply only to a rather limited range of operational situations. This means that whenever there is a small change in conditions, such as the assigned trajectory, required speed or different starting point, the gain settings would have to be readjusted. Furthermore, this system has shown itself to be very sensitive to even minor tweaks to the gain settings. As a result, the adjustments must be carried out using very fine gradations. Therefore, without the help of an established technique for tuning this cascaded PID system, the manual tuning procedure would require plenty of time and effort. Thus, the practical applications of this system are questionable at best.

Due to the obvious limitations of the PID-based system, an alternative adaptive model was developed. The simulation results of the adaptive system were generally very satisfactory and clearly superior to those of the PID model. Also, in contrast to the PID model, optimum results in various test conditions were attained with only very slight adjustments made to the gain settings of the adaptive system. This indicates that the system is versatile and relatively easy to use.

In the validation tests, the good experimental results of the actual prototype confirmed the performance level as predicted by the simulated model. The WMR also performed very well in the widely-cited UMBmark test when compared to published results of other dead-reckoning systems that have undergone the same test. Furthermore, the advantage of using two

sensors over one is clearly demonstrated theoretically and experimentally. This outcome provides a clear answer to Research Question 1.

Despite the excellent results in comparison to similar systems, the tracking accuracy is still unable to match the precision level of the sensors. The sensor data has proved to be remarkable consistent during calibration tests in near-ideal test conditions, i.e. very smooth tracking surface, similar surface texture throughout, steady sensor height due to WMR being fixed in a sturdy frame and slid along without tyres rotating, etc. Even though electrical noise is an unavoidable characteristic of all electronic components, the results indicate that it is not a significant issue here. Thus, it is hard to doubt the reliability or stability of the sensor's readings.

The conditions during regular experimental runs are less uniform or ideal than the short calibration tests. So, the sources and magnitudes of errors are almost certain to be greater than those seen in the calibration tests. It is a well-established fact that the sensors are highly sensitive to changes in operational conditions. They also have a very narrow optimal operational range and may suffer a severe loss of precision when operating outside its limits. Therefore, it is very likely that non-optimal operational conditions are a more significant factor in the loss of tracking accuracy than intrinsic random error emanating from the sensors themselves. This answers Research Question 3.

An essential factor in the accuracy of a WMR's localisation is the determination of its odometry. In the PID-based model, a first-order Euler approximation is used for odometric calculations, while the adaptive model adopted a hybrid geometric approach. In the past, most WMRs used the Euler method because most of them employed a single sensor for tracking. In recent years, more researchers have begun using more than one sensor for their WMRs, and are thus able to adopt a geometric approach for odometry.

In the theoretical analysis, the Euler method is shown to be acceptably accurate when sampling periods are very small, i.e. high sampling frequencies. Also, its accuracy increases as the WMR's turning angle approaches zero. In contrast, the geometric approach is an ideal model that is theoretically exact and not affected by sampling frequency or turning angle. It provides much greater accuracy than Euler-based systems. However, a purely geometric approach has serious limitations when it comes to practical applications.

An in-depth examination of the geometric model reveals several fundamental issues that would greatly affect the accuracy of the odometry when applied to a real system. Firstly, rounding errors occur when trying to avoid divide-by-zero scenarios. Secondly, truncation errors are inevitable as values of system variables approach infinity. Lastly, calculation accuracy is limited by the computer (machine) precision according to the IEEE Standard for

Floating-Point Arithmetic (IEEE 754). Computer precision decreases as numerical magnitude increases. In combination, these factors greatly affect the odometric accuracy of the WMR's localisation process.

These problems have been fully addressed in the theoretical sections of the thesis and a unique hybrid geometric localisation model is proposed. The results of the hardware testing clearly demonstrate the superiority of the hybrid method over the Euler-based approach. Furthermore, the results also show that the hybrid geometric method outperformed other similar systems that adopted the purely geometric approach. This validates the soundness of the new method. The answers to Research Questions 4 and 5 are that the current odometric methods have significant shortcomings due to previously undocumented factors which have now been addressed in this research. However, the possibility cannot be ruled out that there are still some unknown factors that contribute to the loss of tracking accuracy other than purely random errors.

All the recognised issues that affect the tracking accuracy of a WMR have been thoroughly discussed in this thesis and solutions have been proposed for some of them. As mentioned previously, other unknown factors may exist. While every issue contributes to a loss of accuracy to a certain extent, some problems have a far more significant effect than others. Most notably, it is almost impossible for the WMR to keep its sensors operating within the optimal ground clearance range of 0.6 mm on a typical tracking surface. Even the maximum allowed range is only 1.6 mm. Thus, repeated degradation in sensor precision cannot be avoided. This is certainly one of the most significant concerns of all.

The narrow optimal range of the gap between sensor and tracking surface has proven to be the biggest known factor affecting tracking accuracy. In order to alleviate the problem, it is suggested that the sensor be mounted at a higher position and its existing lens be swapped for another with a longer focal length. With this set-up, changes in the sensor elevation due to surface undulations would be proportionally smaller in comparison with the extended focal length. This means that the loss of focus is reduced, and precision can thus be maintained at acceptable levels. This could result in a big improvement in terms of tracking accuracy. The downside of this approach is that the sensor readings would be more affected by any tilting motion of the WMR. Until this modification is implemented, there are still other improvements to tracking accuracy and reliability that can be introduced in the meantime.

In this research, the advantage of using two sensors over one is clearly demonstrated theoretically and experimentally. In addition, if either of the sensors is producing unreliable readings, a new method is proposed to detect the error and make necessary adjustments in the odometric calculations. This provides the tracking control system some form of redundancy. Without this feature, even short spells of reading error would severely affect the

accuracy of the WMR's localisation. Since this approach relies on using one sensor for odometry if the other fails, it cannot fully compensate for the loss of data from the other sensor. Hence, it can only be considered a partially-redundant system. Even though this is not a perfect solution, it helps maintain a certain level of tracking accuracy and thus leads to greater overall reliability of the system. The results of hardware testing clearly show the promise of this method. Research Question 6 is addressed by this partially-redundant technique as well as the proposal for changes in the sensor mounting height and focal length of its lens.

To sum up, it is useful to restate the accomplishments and novelties of this research. This study was able to successfully combine the use of an adaptive controller and a dual optical mouse sensor system to develop an accurate tracking system for a wheeled robot. In the process, several factors were identified that have a significant collective impact on the odometric precision of a purportedly ideal localisation method. These errors have not been documented in existing literature until now. They have been addressed in this thesis and a unique solution has been provided. In addition, a new error-detection and correction method has been proposed to provide partial redundancy for the system in case one of the sensors experiences a temporary data fault. As envisioned, the use of relatively inexpensive off-the-shelf components made it possible to deliver a prototype that is low-cost and requires low processing power, but is still able to perform its task effectively. Finally, even though dead reckoning has fallen out of favour in recent times, this study shows that there is plenty of usefulness left in this navigational approach.

10.1 Future Work

One of the most notable problems encountered when using an optical mouse sensor (OMS) for robot tracking is the susceptibility to reading errors due to fluctuations in the distance between the sensor and tracking surface. The optimal gap size between sensor and surface is typically about 2 mm or so. Thus, even a slight variation in the height of the gap caused by surface imperfections could lead to a significant divergence from the norm in terms of proportion.

If the current sensor lenses were swapped for ones with a longer focal length, the sensors could be mounted at a much higher position. In such a configuration, minor changes in the gap distance will no longer amount to a large proportional deviation. This means that the sensor can maintain focus more easily and its height sensitivity would be reduced. As a result, the sensor would be more tolerant of surface undulations and tracking performance would be greatly improved. This effect has been demonstrated previously (Jackson et al. 2007) where an OMS mounted on the bumper of a standard road vehicle produced an average error rate of just 1.07% over a distance of 22.25 m.

The advantage of having the sensors relocated to a higher position is clear. However, there is an obvious drawback. Distortions in the sensors' readings caused by tilting would be magnified as a result. In order to alleviate this problem, the sensors could be mounted on a gyro-stabilised gimbal system. This would ensure that the sensors are always pointed vertically downwards. Unfortunately, this option would add significant expense and complexity to the system. For low-cost platforms, this would not be feasible. A more realistic approach would be to find the optimum sensor mounting height where there is a balanced trade-off between sensitivity to gap distance and vehicle tilting.

A potential enhancement to the current prototype is an obstacle avoidance system. Since the tracking sensors face downwards and cannot detect any obstruction in its path, forward-looking sensors have to be added, such as infrared or ultrasonic ones. Naturally, the necessary algorithms would have to be added to the main control system.

Aside from looking at ways to improve the existing system, potential applications in other related research areas should also be investigated. One of them is the use of the OMS tracking system as a localisation subsystem for a mapping robot. For systems employing Simultaneous Localisation and Mapping (SLAM) techniques, their localisation accuracy is nowhere near the level provided by systems using OMS. This is mostly due to methodological or technological limitations. Once the mapping phase is complete, the OMS tracking can be turned off or kept running in the background for redundancy purposes.

References

Agilent Technologies (2004). Agilent 20x Surface Tracking Improvement for Laser-Based Mice. Santa Clara, CA, USA, Agilent Technologies.

Åström, K. J., Borisson, U., et al. (1977). Theory and Applications of Self-Tuning Regulators. *Automatica* 13(5): 457-476.

Åström, K. J. and Hägglund, T. (1995). PID Controllers - Theory, Design, and Tuning (2nd Edition). ISA.

Åström, K. J. and Wittenmark, B. (1973). On Self Tuning Regulators. *Automatica* 9(2): 185-199.

Atmel (2008). AVR151: Setup and use of the SPI on tinyAVR and megaAVR device (Revision C). Atmel Corporation. From: http://www.atmel.com/Images/doc2585.pdf.

Atmel (2009). AVR311: Using the TWI module as I2C slave on tinyAVR and megaAVR devices (Revision D). Atmel Corporation. From: http://www.atmel.com/Images/doc2565.pdf.

Atmel (2010). AVR315: Using the TWI module as I2C master on tinyAVR and megaAVR devices (Revision C). Atmel Corporation. From: http://www.atmel.com/Images/doc2564.pdf.

Atmel (2011a). AT91SAM ARM-based Flash MCU (Revision L). Atmel Corporation. From: http://www.atmel.com/Images/doc6175.pdf.

Atmel (2011b). ATmega32(L) Complete (Revision Q). Atmel Corporation. From: http://www.atmel.com/Images/doc2503.pdf.

Avago (2009a). ADNS-2051 Optical Mouse Sensor - Data Sheet. Avago Technologies/Agilent Technologies, 2002. From: http://www.avagotech.com/docs/AV02-1364EN.

Avago (2009b). ADNS-6010 Laser Mouse Sensor - Data Sheet. Avago Technologies/Agilent Technologies, 2005. From: http://www.avagotech.com/docs/AV02-1410EN.

Avago (2011). ADNS-9500 LaserStream Gaming Sensor - Data Sheet. Avago Technologies, 2009. From: http://www.avagotech.com/docs/AV02-1726EN.

Avago (2012). ADNS-9800 LaserStream Gaming Sensor - Data Sheet. Avago Technologies, 2012. From: http://www.avagotech.com/docs/AV02-2998EN.

Beauchemin, S. S. and Barron, J. L. (1995). The Computation of Optical Flow. *ACM Comput. Surv.* 27(3): 433-466.

Beckman, B. (1991). The Physics of Racing, Part 7: The Traction Budget. The Physics of Racing. From: http://phors.locost7.info/phors07.htm.

Bennett, S. (2001). The Past of PID Controllers. *Annual Reviews in Control* 25(0): 43-53.

Bilenca, A. Laser Speckle. S. imaging2.gif, Wellman Center for Photomedicine, Massachusetts General Hospital.

Blais, F. (2004). Review of 20 Years of Range Sensor Development. *Journal of Electronic Imaging* 13(1): 231-243.

Bobtsov, A. A., Pyrkin, A. A., et al. (2011). Using of LEGO Mindstorms NXT Technology for Teaching of Basics of Adaptive Control Theory. 18th International Federation of Automatic Control (IFAC) World Congress. Milan, Italy, International Federation of Automatic Control (IFAC).

Bonarini, A., Matteucci, M., et al. (2004). Dead Reckoning for Mobile Robots Using Two Optical Mice. First International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004). H. Araújo, A. Vieira, J. Braz, B. Encarnação and M. Carvalho. Setúbal, Portugal, INSTICC Press**:** 87-94.

Bonarini, A., Matteucci, M., et al. (2005). Automatic Error Detection and Reduction for an Odometric Sensor based on Two Optical Mice. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA 2005)*.

Borenstein, J., Everett, H. R., et al. (1997). Mobile Robot Positioning: Sensors and Techniques. *Journal of Robotic Systems* 14(4): 231-249.

Borenstein, J. and Feng, L. (1995). UMBmark - A Benchmark Test for Measuring Odometry Errors in Mobile Robots**:** 124.

Borenstein, J. and Feng, L. (1996). Measurement and Correction of Systematic Odometry Errors in Mobile Robots. *IEEE Transactions on Robotics and Automation* 12(6): 12.

Boyden, F. D. and Velinsky, S. A. (1994a). Dynamic Modeling of Wheeled Mobile Robots for High Load Applications. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, CA.

Boyden, F. D. and Velinsky, S. A. (1994b). Limitations of Kinematic Models for Wheeled Mobile Robots. Advances in Robot Kinematics and Computational Geometry. A. J. Lenarcic and B. B. Ravani. Kluwer Academic Publishers, Amsterdam, the Netherlands: 151-160.

Campion, G., Bastin, G., et al. (1996). Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots. *IEEE Transactions on Robotics and Automation* 12(1): 47-62.

Center for Advanced Spatial Technologies. Uneven beam focus spot, University of Arkansas.

Chikamasa, T. (2007). LEGO MINDSTORMS NXT system architecture. NXT_system_architecture.jpg, nxtOSEK.

Cruz-Martín, A., Fernández-Madrigal, J. A., et al. (2012). A LEGO Mindstorms NXT Approach for Teaching at Data Acquisition, Control Systems Engineering and Real-Time Systems Undergraduate Courses. *Computers & Education* 59(3): 974-988.

Cuéllar, M. P. and Pegalajar, M. C. (2011). Design and Implementation of Intelligent Systems with LEGO Mindstorms for Undergraduate Computer Engineers. *Computer Applications in Engineering Education*: n/a-n/a.

Curless, B. (1999). From Range Scans to 3D Models. *SIGGRAPH Computer Graphics* 33(4): 38-41.

d'Andrea-Novel, B., Bastin, G., et al. (1991). Modelling and Control of Non-Holonomic Wheeled Mobile Robots. *Proceedings of the IEEE International Conference on Robotics and Automation, 1991*.

Dalsager, H. W., Esbensen, T., et al. (2006). Camera Controlled Robot. Faculty of Engineering, Science and Medicine, Department of Electronic Systems. Aalborg, Denmark, Aalborg University**:** 124.

Dille, M., Grocholsky, B., et al. (2010). Outdoor Downward-Facing Optical Flow Odometry with Commodity Sensors

Field and Service Robotics. A. Howard, K. Iagnemma and A. Kelly. Springer Berlin / Heidelberg. 62: 183-193.

Donati, S. (2004). Electro-Optical Instrumentation: Sensing and Measuring with Lasers, 1. Prentice Hall.

Dschwen (2006). Constant Fraction. Constant_fraction_1.svg.png, Wikimedia Commons.

Dumont, G. A. and Huzmezan, M. (2002). Concepts, Methods and Techniques in Adaptive Control. *American Control Conference, 2002. Proceedings of the 2002*.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE* 13(2): 99-110.

Edlén, B. (1966). The Refractive Index of Air. *Metrologia* 2: 71-80.

Everett, H. R. (1995). Sensors for Mobile Robots: Theory and Application. A K Peters, Ltd.

Fontes, F. A. C. C. and Magni, L. (2004). A Generalization of Barbalat's Lemma with Applications to Robust Model Predictive Control. Proceedings of the Sixteenth International Symposium on Mathematical Theory of Networks and Systems (MTNS 2004). Louvain, Belgium.

Freeman, R. A. and Kokotović, P. (1996). Robust Nonlinear Control Design State-Space and Lyapunov Techniques. Birkhäuser, Boston.

Gasperi, M. and Hurbain, P. E. (2009). Extreme NXT: Extending the LEGO MINDSTORMS NXT to the Next Level, Second Edition. Apress, Inc., New York, NY 10013, USA.

Gholipour, A. and Yazdanpanah, M. J. (2003). Dynamic Tracking Control of Nonholonomic Mobile Robot with Model Reference Adaptation for Uncertain Parameters. 38th European Control Conference (ECC 2003) Cambridge, U.K.

Goldberg, D. (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys* 23(1): 5-48.

Gonçalves, J., Lima, J., et al. (2009). Realistic Simulation of a Lego Mindstorms NXT Based Robot. *Third IEEE Multi-conference on Systems and Control (MSC 2009) consisting of 18th IEEE International Conference on Control Applications (CCA) & 24th IEEE International Symposium on Intelligent Control (ISIC)*, Saint Petersburg, Russia.

Grega, W. and Pilat, A. (2008). Real-Time Control Teaching Using LEGO MINDSTORMS NXT Robot. *International Multiconference on Computer Science and Information Technology (IMCSIT 2008)*.

Gustafson, E. H., Lollini, C. T., et al. (2005). Swarm Technology for Search and Rescue Through Multi-Sensor Multi-Viewpoint Target Identification. *Proceedings of the 37th Southeastern Symposium on System Theory, 2005 (SSST '05)*.
Hancock, J. A. (1999). Laser Intensity-Based Obstacle Detection and Tracking, Carnegie Mellon University, The Robotics Institute.
Hebert, M. and Krotkov, E. (1991). 3-D Measurements from Imaging Laser Radars: How Good are They? *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, 1991, 'Intelligence for Mechanical Systems'*.
Hong, D., Velinsky, S. A., et al. (1999). Verification of a Wheeled Mobile Robot Dynamic Model and Control Ramifications. *Journal of Dynamic Systems, Measurement and Control - Transactions of the ASME* 121(1): 58-63.
Hurbain, P. E. (2007a). LEGO 9V Technic Motors compared characteristics. From: http://www.philohome.com/motors/motorcomp.htm.
Hurbain, P. E. (2007b, Nov 2008). NXT motor internals. From: http://www.philohome.com/nxtmotor/nxtmotor.htm.
Hyun, D., Yang, H. S., et al. (2009). Differential optical navigation sensor for mobile robots. *Sensors and Actuators A: Physical* 156(2): 296-301.
Ioannou, P. A. and Sun, J. (1996). Robust Adaptive Control. Prentice Hall PTR, Upper Saddle River, New Jersey, USA.
Jackson, J. D., Callahan, D. W., et al. (2007). A Rationale for the Use of Optical Mice Chips for Economic and Accurate Vehicle Tracking. *Third IEEE International Conference on Automation Science and Engineering (CASE 2007)*.
JED (2005). AVR200 Schematic (colour). JED Microprocessors. From: http://www.jedmicro.com.au/200v1_Schematic_colour.pdf.
JED (2008, 23 Nov). JED AVR200 Single Board Computer using the ATmega32 CPU. From: http://www.jedmicro.com.au/avr200.htm.
Kane, T. R. and Levinson, D. A. (1983). The use of Kane's Dynamical Equations in Robotics. *International Journal of Robotics Research* 2: 3-21.
Khalil, H. (2002). Nonlinear Systems, 3rd Ed. Prentice Hall.
Kuo, B. C. (1995). Automatic Control Systems, 7. Prentice-Hall, Upper Saddle River, NJ.
Landau, I. D. (2003). Controls, Adaptive Systems. Encyclopedia of Physical Science and Technology (Third Edition). R. A. Meyers. Academic Press, New York: 649-658.
Landau, I. D., Lozano, R., et al. (2011). Adaptive Control - Algorithms, Analysis and Applications, 2nd Ed. Springer.
Langer, D. and Thorpe, C. (1992). Sonar Based Outdoor Vehicle Navigation And Collision Avoidance. *Proceedings of the 1992 lEEE/RSJ International Conference on Intelligent Robots and Systems*.
Lee, Y., Park, S., et al. (1998). PID Controller Tuning To Obtain Desired Closed Loop Responses for Cascade Control Systems. *Industrial & Engineering Chemistry Research* 37(5): 1859-1865.
Lego (2006). LEGO MINDSTORMS NXT Hardware Developer Kit (Version 1.00). Lego Group. From: http://mindstorms.lego.com/en-us/support/files/default.aspx.
Leonard, J. J. and Durrant-Whyte, H. F. (1991). Simultaneous Map Building and Localization for an Autonomous Mobile Robot. *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, 1991 (IROS '91), 'Intelligence for Mechanical Systems'*, Osaka, Japan.
Leva, A. and Marinelli, A. (2009). Comparative Analysis of Some Approaches to the Autotuning of Cascade Controls. *Industrial & Engineering Chemistry Research* 48(12): 5708-5718.
Logitech (2004). Laser Technology Brief, Logitech.
Martinec, D. and Hurak, Z. (2011). Vehicular Platooning Experiments with LEGO MINDSTORMS NXT. *Control Applications (CCA), 2011 IEEE International Conference on*.
Maxim (2011, 14 Jan). Motor controller with feed-forward for Lego NXT. From: http://nxt-unroller.blogspot.com/2011/01/motor-controller-with-feed-forward-for.html.
Minoni, U. and Signorini, A. (2006). Low-Cost Optical Motion Sensors: An Experimental Characterization. *Sensors and Actuators A: Physical* 128(2): 402-408.
Monteiro, L. S., Moore, T., et al. (2005). What is the accuracy of DGPS? *The Journal of Navigation* 58(2): 207-225.

Montemerlo, M., Thrun, S., et al. (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. Proceedings of the Eighteenth AAAI National Conference on Artificial Intelligence, 2002. Edmonton, Alberta, Canada, American Association for Artificial Intelligence.

Montemerlo, M., Thrun, S., et al. (2003). FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)*, Acapulco, Mexico, Morgan Kaufmann Publishers, San Francisco, USA.

Murray, R. M., Li, Z., et al. (1994). A Mathematical Introduction to Robotic Manipulation. CRC Press.

Nukulwuthiopas, W., Laowattana, S., et al. (2002). Dynamic Modeling of a One-Wheel Robot by Using Kane's Method. *IEEE International Conference on Industrial Technology, 2002*.

NXP (2012). I2C-Bus Specification and User Manual (Rev. 4). NXP Semiconductors. From: http://www.nxp.com/documents/user_manual/UM10204.pdf.

O'Hara, K. J. and Kay, J. S. (2003). Investigating Open Source Software and Educational Robotics. *Journal of Computing Sciences in Colleges* 18(3): 8-16.

Oriolo, G., De Luca, A., et al. (2002). WMR Control via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation. *IEEE Transactions on Control Systems Technology* 10(6): 835-852.

Palacin, J., Valgañon, I., et al. (2006). The Optical Mouse for Indoor Mobile Robot Odometry Measurement. *Sensors and Actuators A: Physical* 126(1): 141-147.

Park, H. R., Hyun, D. J., et al. (2009). A Dead Reckoning Sensor System and a Tracking Algorithm for Mobile Robot. *ICCAS-SICE, 2009*.

Parks, P. C. (1992). A. M. Lyapunov's Stability Theory - 100 Years On. *IMA Journal of Mathematical Control and Information* 9(4): 275-303.

Pentland, A. P. (1987). A New Sense for Depth of Field. *IEEE Transactions Pattern Analysis and Machine Intelligence* 9(4): 523-531.

Pieper, R., Cooper, A., et al. (1995). Dual Baseline Triangulation. *Proceedings of the Twenty-Seventh Southeastern Symposium on System Theory, 1995*.

Popov, V. M. (1973). Hyperstability of Control Systems, Revised Ed. Editura Academiei Bucarest / Springer-Verlag.

Pourboghrat, F. and Karlsson, M. P. (2002). Adaptive Control of Dynamic Mobile Robots with Nonholonomic Constraints. *Computers & Electrical Engineering* 28(4): 241-253.

Sadasivarao, M. V. and Chidambaram, M. (2006). PID Controller Tuning of Cascade Control Systems Using Genetic Algorithm. *Journal of the Indian Institute of Science* 86(4): 343-354.

Saha, S. K. and Angeles, J. (1989). Kinematics and Dynamics of a Three-Wheeled 2-DOF AGV. *Proceedings of the IEEE International Conference on Robotics and Automation, 1989*.

Sarkar, N., Yun, X., et al. (1993). Dynamic Path Following: A New Control Algorithm for Mobile Robots. *Proceedings of the 32nd IEEE Conference on Decision and Control, 1993*, San Antonio, Texas, USA.

Sarkar, N., Yun, X., et al. (1994). Control of Mechanical Systems With Rolling Constraints: Application to Dynamic Control of Mobile Robots. *The International Journal of Robotics Research* 13(1): 55-69.

Sastry, S. and Bodson, M. (1994). Adaptive Control: Stability, Convergence, and Robustness Prentice Hall.

Schechner, Y. Y. and Kiryati, N. (1998). Depth from Defocus vs. Stereo: How Different Really Are They? *Proceedings of the Fourteenth International Conference on Pattern Recognition, 1998*.

Schoell, E. (2004). AVR200 Board technical description. JED Microprocessors. From: http://www.jedmicro.com.au/AVR200%20Project%20Single%20Board%20Computer.pdf.

Sekimori, D. and Miyazaki, F. (2005). Self-Localization for Indoor Mobile Robots Based on Optical Mouse Sensor Values and Simple Global Camera Information. *2005 IEEE International Conference on Robotics and Biomimetics (ROBIO)*.

Sekimori, D. and Miyazaki, F. (2007). Precise Dead-Reckoning for Mobile Robots using Multiple Optical Mouse Sensors Informatics in Control. Automation and Robotics II. J. Filipe, J.-L. Ferrier, J. A. Cetto and M. Carvalho. Springer Netherlands: 145-151.

Seyr, M. and Jakubek, S. (2005). Mobile Robot Predictive Trajectory Tracking. *Proceedings of the Second International Conference on Informatics in Control (ICINCO 2005), 'Automation and Robotics'*, Barcelona, Spain.
Shaw, A. (2011, 20 Feb). i2c.c: Added a higher speed i2c operating mode (125KHz rather than 9.6KHz) (Revision 4241). From: http://lejos.svn.sourceforge.net/viewvc/lejos/trunk/nxtvm/platform/nxt/i2c.c?revision=4241&view=markup.
Siegwart, R. and Nourbakhsh, I. R. (2004). Autonomous Mobile Robots. MIT Press.
Silva, V., Santos, F., et al. (2002). A Robust Self-Localization System for a Small Mobile Autonomous Robot. Proceedings of the 3rd ANIRob/IEEE Int. Symposium on Robotics and Automation (ISRA 2002). Toluca, Mexico.
Singh, S. P. N. and Waldron, K. J. (2004). Design and Evaluation of an Integrated Planar Localization Method for Desktop Robotics. *Proceedings of the IEEE International Conference on Robotics and Automation, 2004 (ICRA 2004)*, New Orleans, LA, USA.
Slotine, J.-J. and Li, W. (1991). Applied Nonlinear Control. Prentice Hall.
Spong, M. W., Hutchinson, S., et al. (2006). Robot Modeling and Control. John Wiley & Sons, Hoboken, NJ.
Tan, K. K., Wang, Q.-G., et al. (1999). Advances in PID Control. Springer-Verlag.
Tan, W., Liu, J., et al. (2006). Comparison of Some Well-Known PID Tuning Formulas. *Computers & Chemical Engineering* 30(9): 1416-1423.
Teo, C. M. (2006). Understanding Optical Mice. Avago Technologies. From: www.avagotech.com/docs/AV02-1265EN.
Thakoor, S., Morookian, J. M., et al. (2004). BEES: Exploring Mars with Bioinspired Technologies. *Computer* 37(9): 38-47.
Thanjavur, K. and Rajagopalan, R. (1997). Ease of Dynamic Modelling of Wheeled Mobile Robots (WMRs) using Kane's Approach. *Proceedings of the IEEE International Conference on Robotics and Automation, 1997*.
Tuley, J., Vandapel, N., et al. (2005). Analysis and Removal of Artifacts in 3-D LADAR Data. *Proceedings of the IEEE International Conference on Robotics and Automation, 2005*: 2203-2210.
USAF (2008). GPS Standard Positioning Service (SPS) Performance Standard 4th Edition. United States Air Force Global Positioning Systems Directorate. From: http://www.gps.gov/technical/ps/.
Watanabe, R. (2007). NXT Motor. From: http://web.mac.com/ryo_watanabe/iWeb/Ryo's Holiday/NXT Motor.html.
Yamamoto, Y. (2009a, 28 Apr). NXT Ballbot (Self-Balancing Robot On A Ball) Controller Design. From: http://www.mathworks.com/matlabcentral/fileexchange/23931.
Yamamoto, Y. (2009b, 1 May). NXTway-GS (Self-Balancing Two-Wheeled Robot) Controller Design. From: http://www.mathworks.com/matlabcentral/fileexchange/19147.
Yatsenko, L. P., Shore, B. W., et al. (2004). Ranging and Interferometry with a Frequency Shifted Feedback Laser. *Optics Communications* 242(4–6): 581-598.
Yoshida, K. and Hirose, S. (1988). Laser Triangulation Range Finder Available under Direct Sunlight. *Proceedings of the IEEE International Conference on Robotics and Automation, 1988*.
Zhang, Y., Chung, J. H., et al. (2003). Variable Structure Control of a Differentially Steered Wheeled Mobile Robot. *Journal of Intelligent and Robotic Systems* 36(3): 301-314.
Zhang, Y., Hong, D., et al. (1998). Dynamic Model Based Robust Tracking Control of a Differentially Steered Wheeled Mobile Robot. *Proceedings of the American Control Conference, 1998*.
Zhao, Y. and BeMent, S. L. (1992). Kinematics, Dynamics and Control of Wheeled Mobile Robots. *Proceedings of the IEEE International Conference on Robotics and Automation, 1992*.
Zhou, K. and Doyle, J. C. (1998). Essentials Of Robust Control. Prentice Hall.

Appendix A

A.1 Enlarged version of Adaptive Model of WMR (Fig. 7.1)

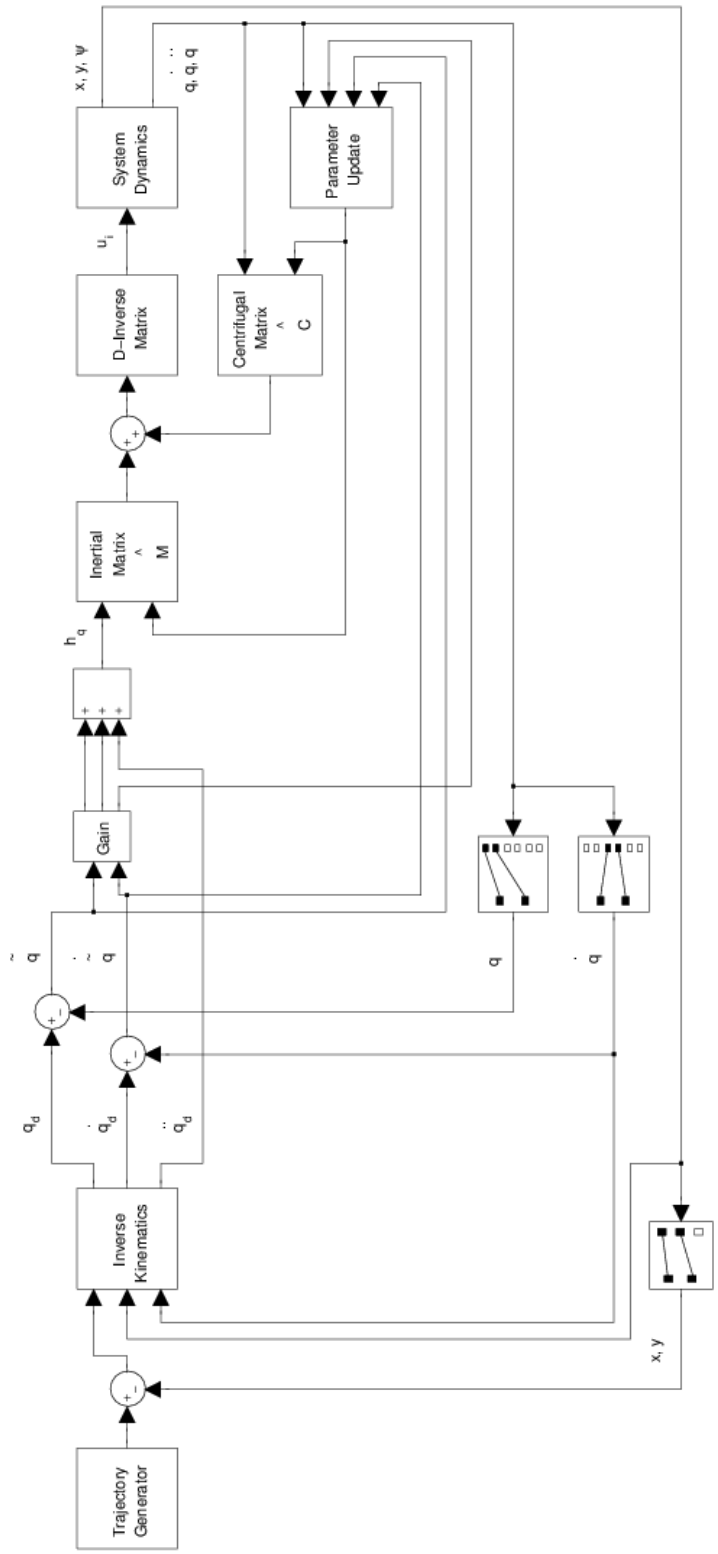


Fig. A.1 Adaptive Tracking Control System of the WMR

A.2 Embedded MATLAB Code for Trajectory Generator

Trajectory Generator function block as seen in Fig. 4.2 of Section 4.3.1 and Fig.7.2 of Section 7.3.1

```
function [x, y] = ref_path(t)

%Reference trajectory

% **** Straight line ****
x = 0.2 * t;
y = 0;
% y = 0.5;
% y = 0.5 * x + 0.5;


%{
% **** Sinusoidal waveform ****
x = 0.2 * t;
y = sin(x);
% y = 2 * sin(2 * x);
%}

%{
% **** Triangular waveform ****
x = 0.05 * t;
period = 2;
phase = 0;
ratio = 0.5;
amp = 2;
u1 = rem(x + phase * period / (2 * pi) + period, period);
y = (u1 <= period * ratio) .* u1 * amp / (period * ratio) + (u1 > period * ratio)...
    .* (period - u1) * amp / (period * (1 - ratio));
%}

%{
% **** Circular waveform ****
% Note: Ratio of diameter to chord is 2/pi
x_offset = 2;
y_offset = 0;
r = 2;
d = 2 * r;

x = mod((0.12 * t), (2 * d)) + (x_offset - r);
if x > (d + x_offset - r)
    x = 2 * (d + x_offset - r) - x;
    y = y_offset - sqrt(r^2 - (x - x_offset)^2);
else
    y = y_offset + sqrt(r^2 - (x - x_offset)^2);
end
%}

%{
% **** Square trajectory ****
t_factor = 0.2;
length = 4;
t_length = length / t_factor;
x_offset = 0;
y_offset = 0;

if (t <= t_length)
    x = t_factor * t + x_offset;
    y = 0 + y_offset;
elseif ((t > t_length) && (t <= (2 * t_length)))
    x = length + x_offset;
    y = t_factor * (t - t_length) + y_offset;
elseif ((t > 2 * t_length) && (t <= 3 * t_length))
    x = length - t_factor * (t - 2 * t_length) + x_offset;
    y = length + y_offset;
elseif ((t > 3 * t_length) && (t <= 4 * t_length))
```

```
    x = 0 + x_offset;
    y = length - t_factor * (t - 3 * t_length) + y_offset;
else
    x = 0 + x_offset;
    y = 0 + y_offset;
end
%}

%{
% **** Square trajectory with chamfer ****
t_factor = 0.2;
length = 4;
t_length = length / t_factor;
x_offset = 0;
y_offset = 0;
cnr_offset = 0.04;
t_cnr = cnr_offset / t_factor;

if (t <= (t_length - t_cnr))
    x = t_factor * t + x_offset;
    y = 0 + y_offset;
elseif ((t > (t_length - t_cnr)) && (t <= t_length))
    x = t_factor * t + x_offset;
    % y = x - (t_factor * (t_length - t_cnr) + y_offset);
    y = x - length + cnr_offset + y_offset;
elseif ((t > t_length) && (t <= (2 * (t_length - t_cnr))))
    x = length + x_offset;
    % y = t_factor * (t - t_length + t_cnr) + y_offset;
    y = t_factor * t - length + cnr_offset + y_offset;
elseif ((t > (2 * (t_length - t_cnr))) && (t <= (2 * t_length - t_cnr)))
    % y = t_factor * (t - t_length + t_cnr) + y_offset;
    y = t_factor * t - length + cnr_offset + y_offset;
    x = length - (y - (length - cnr_offset + y_offset)) + x_offset;
elseif ((t > (2 * t_length - t_cnr)) && (t <= (3 * (t_length - t_cnr))))
    x = length - cnr_offset - (t_factor * t - (2 * length - cnr_offset)) + x_offset;
    y = length + y_offset;
elseif ((t > (3 * (t_length - t_cnr))) && (t <= (t_length + 2 * (t_length - t_cnr))))
    x = length - cnr_offset - (t_factor * t - (2 * length - cnr_offset)) + x_offset;
    y = x + (length - cnr_offset) + y_offset;
elseif ((t > (t_length + 2 * (t_length - t_cnr))) && (t <= (t_length + 3 * (t_length...
    - t_cnr))))
    x = 0 + x_offset;
    y = length - cnr_offset - (t_factor * t - (3 * length - 2 * cnr_offset)) + y_offset;
else
    x = 0 + x_offset;
    y = 0 + y_offset;
end
%}
```

A.3 Embedded MATLAB Code for About-Turn Algorithm

About-Turn function block as seen in Fig. 4.4 of Sections 4.3.3 and 4.3.3.1

```
function [w_adj, theta] = turn(w, err, pos, cons, wmax)
% Algorithm for resolving reverse motion and saturation

% Initialisation of variables
xq = 0;
yq = 0;
xb = 0;
yb = 0;
xl = 0;
yl = 0;
xr = 0;
yr = 0;
xp = 0;
yp = 0;
x = 0;
y = 0;
coeff1 = 0;
coeff2 = 0;
coeff3 = 0;
eqn = [0, 0, 0];
xt = [0 + 0i; 0 + 0i];
x1 = 0;
y1 = 0;
x2 = 0;
y2 = 0;
ch1 = 0;
ch2 = 0;
xtan1 = 0;
ytan1 = 0;
xtan2 = 0;
ytan2 = 0;
norm1 = [0; 0];
norm2 = [0; 0];
norm3 = [0; 0];
unit_norm1 = [0; 0];
unit_norm2 = [0; 0];
unit_norm3 = [0; 0];
ang_normals1 = 0;
ang_normals2 = 0;
ang_normals3 = 0;
wheel_vect = [0; 0];
unit_wheel_vect = [0; 0];
ang_disp = 0;
circ_disp = 0;
theta_l = 0;
theta_r = 0;
omega_l = 0;
omega_r = 0;

% Wheel angular velocity [rad/s]
% omega_l = w(1);
% omega_r = w(2);

% Positional error [m]
del_x = err(1);
del_y = err(2);

% Current sensor position [m]
xa = pos(1);
ya = pos(2);
```

```
% Current orientation [rad]
psi = pos(3);

% Wheel radius [m]
r = cons(1);

% Distance between wheels [m]
d = cons(2);

% Distance between sensor and centre of wheel baseline [m]
a = cons(3);

% Maximum angular velocity of motor [rad/s]
omega_max = wmax;

% **** Mathematical aide-memoire [Start] ****

% Equation of circle with centre at (xp, yp)
% (x - xp)^2 + (y - yp)^2 = rs^2
% y = yp + sqrt(rs^2 - (x - xp)^2)

% Derivative
% dy/dx = -(x -xp) / (y - yp)

% Arc length
% s = int(sqrt(1 + (dy/dx)^2), x)

% Equation of tangent
% y - y1 = dy/dx * (x - x1)

% **** Mathematical aide-memoire [End] ****

if (w(1) < 0) && (w(2) < 0)
    % Turns WMR around using one wheel as a stationary pivot

    % Radius of circle (distance between pivot wheel & baseline centre [m]
    rs = d / 2;

    % Reference sensor position [m]
    xq = xa + del_x;
    yq = ya + del_y;

    % Determination of position of wheel baseline centre [m]
    % tan(psi) = (ya - yb) / (xa - xb)
    % ((xa - xb)^2 + (ya - yb)^2 = a^2
    xb = xa - a * cos(psi);
    yb = ya - a * sin(psi);

    % Determination of positions of wheels [m]
    xl = xb + rs * cos(psi + pi /2);
    yl = yb + rs * sin(psi + pi /2);
    xr = xb + rs * cos(psi - pi /2);
    yr = yb + rs * sin(psi - pi /2);

    % Determination of positions of wheels [m] (Alternative)
    % if (psi >= 0) && (psi < pi / 2)
    %     xl = xb - rs * (-cos(psi + pi / 2));
    %     yl = yb + rs * (sin(psi + pi / 2));
    %     xr = xb + rs * (cos(psi - pi / 2));
    %     yr = yb - rs * (-sin(psi - pi / 2));
    % elseif (psi >= pi / 2) && (psi < pi)
    %     xl = xb - rs * (-cos(psi + pi / 2));
    %     yl = yb - rs * (-sin(psi + pi / 2));
    %     xr = xb + rs * (cos(psi - pi / 2));
    %     yr = yb + rs * (sin(psi - pi / 2));
```

```
% elseif (psi >= pi) && (psi < 3 / 2 * pi)
%     xl = xb + rs * (cos(psi + pi / 2));
%     yl = yb - rs * (-sin(psi + pi / 2));
%     xr = xb - rs * (-cos(psi - pi / 2));
%     yr = yb + rs * (sin(psi - pi / 2));
% else
%     xl = xb + rs * (cos(psi + pi / 2));
%     yl = yb + rs * (sin(psi + pi / 2));
%     xr = xb - rs * (-cos(psi - pi / 2));
%     yr = yb - rs * (-sin(psi - pi / 2));
% end

% Determination of pivoting wheel
if abs(w(1)) <= abs(w(2))
    xp = xl;
    yp = yl;
else
    xp = xr;
    yp = yr;
end

% Points of intersection of tangent and circle [m]
x = xq;
y = yq;

co1 = (x - xp)^2 + (y - yp)^2;
co2 = -2 * rs^2 * (x - xp) - 2 * xp * (x - xp)^2 - 2 * xp * (y - yp)^2;
co3 = xp^2 * ((x - xp)^2 + (y - yp)^2) + rs^2 * (2 * xp * (x - xp)...
    - (y - yp)^2) + rs^4;

eqn = [co1, co2, co3];
xt = roots(eqn);
x1 = real(xt(1));
x2 = real(xt(2));
y1 = yp + sqrt(rs^2 - (x1 - xp)^2);
y2 = yp + sqrt(rs^2 - (x2 - xp)^2);

% Chord length [m]
ch1 = sqrt((x1 - xb)^2 + (y1 - yb)^2);
ch2 = sqrt((x2 - xb)^2 + (y2 - yb)^2);

% Determination of target tangent point
if ch1 > ch2
    xtan1 = x1;
    ytan1 = y1;
    xtan2 = x2;
    ytan2 = y2;
else
    xtan1 = x2;
    ytan1 = y2;
    xtan2 = x1;
    ytan2 = y1;
end

% Normal vectors from centre of circle to tangent points
norm1 = [(xtan1 - xp); (ytan1 - yp)];
norm2 = [(xtan2 - xp); (ytan2 - yp)];

% Angle between normal vectors (the larger angle > 180 deg)
% Note: The complex function is used for fixing MATLAB precision errors
ang_normals1 = abs(real(acos(complex(dot(norm1, norm2) ./...
    (norm(norm1) .* norm(norm2))))));

ang_normals1 = 2 * pi - ang_normals1;

% Normal vector bisecting angle between two other normal vectors
```

```
    unit_norm1 = norm1 ./ (norm(norm1));
    unit_norm2 = norm2 ./ (norm(norm2));
    norm3 = -(unit_norm1 + unit_norm2);
    unit_norm3 = norm3 ./ (norm(norm3));

    % Vector along wheel baseline (orthogonal to orientation)
    wheel_vect = [(xb - xp); (yb - yp)];
    unit_wheel_vect = wheel_vect ./ (norm(wheel_vect));

    % Angle between first normal vector and bisecting normal vector
    ang_normals2 = ang_normals1 / 2;

    % Angle between wheel vector and bisecting normal vector
    % (Note: Both are unit vectors, so no need to divide by magnitude)
    ang_normals3 = abs(acos(dot(unit_wheel_vect, unit_norm3)));

    % Total angular displacement of sensor [rad]
    ang_disp = ang_normals2 + ang_normals3;

    % Circumferential displacement of outer wheel [m]
    circ_disp = d * ang_disp;

    % Angular displacement of wheels [rad]
    if abs(w(1)) <= abs(w(2))
        theta_l = 0;
        theta_r = circ_disp / r;
    else
        theta_l = circ_disp / r;
        theta_r = 0;
    end

    % Angular velocity of wheels (calculated outside function)
    omega_l = 0;
    omega_r = 0;

elseif (w(1) < 0) && (w(2) >= 0)
    yaw_vel = r / d * (w(2) - w(1));
    omega_l = 0;
    omega_r = omega_l + d * yaw_vel / r;
    if omega_r > omega_max
        omega_r = omega_max;
    end
    theta_l = 0;    % Angular displacement of wheel not calculated here
    theta_r = 0;    % Angular displacement of wheel not calculated here
elseif (w(1) >= 0) && (w(2) < 0)
    yaw_vel = r / d * (w(2) - w(1));
    omega_r = 0;
    omega_l = omega_r - d * yaw_vel / r;
    if omega_l > omega_max
        omega_l = omega_max;
    end
    theta_l = 0;    % Angular displacement of wheel not calculated here
    theta_r = 0;    % Angular displacement of wheel not calculated here
elseif (w(1) >= 0) && (w(2) >= 0)
    if (w(1) > omega_max) || (w(2) > omega_max)
        yaw_vel = r / d * (w(2) - w(1));
        if w(1) <= w(2)
            omega_r = omega_max;
            omega_l = omega_r - d * yaw_vel / r;
        else
            omega_l = omega_max;
            omega_r = omega_l + d * yaw_vel / r;
        end
    else
        omega_l = w(1);
        omega_r = w(2);
```

```
    end
    theta_l = 0;    % Angular displacement of wheel not calculated here
    theta_r = 0;    % Angular displacement of wheel not calculated here
end

% Output
w_adj = [omega_l, omega_r];
theta = [theta_l, theta_r];
```

A.4 Simulink S-Function Code for Symmetric Positive Definite Matrix

Symmetric Positive Definite Matrix function block as seen in Fig. 7.8 of Sections 7.3.7 and 7.3.7.3

Note: Simulink code supplied by MathWorks and modified by Loren Yeo for the project

```
function spd_mat(block)
% Stability analysis using Lyapunov function candidate
% Determination of a symmetric positive definite (SPD) matrix
% Default Simulink S-Function code by MathWorks modified by Loren Yeo where stated

%%
%% The setup method is used to setup the basic attributes of the
%% S-function such as ports, parameters, etc. Do not add any other
%% calls to the main body of the function.
%%
setup(block);

%endfunction

%% Function: setup ===================================================
%% Abstract:
%%   Set up the S-function block's basic characteristics such as:
%%   - Input ports
%%   - Output ports
%%   - Dialog parameters
%%   - Options
%%
%%   Required         : Yes
%%   C-Mex counterpart: mdlInitializeSizes
%%
function setup(block)

% Register number of ports
block.NumInputPorts  = 1;
block.NumOutputPorts = 1;

% Setup port properties to be inherited or dynamic
block.SetPreCompInpPortInfoToDynamic;
block.SetPreCompOutPortInfoToDynamic;

% Override input port properties
block.InputPort(1).Dimensions        = 2;
block.InputPort(1).DatatypeID  = 0;  % double
block.InputPort(1).Complexity  = 'Real';
block.InputPort(1).DirectFeedthrough = true;

% Override output port properties
block.OutputPort(1).Dimensions       = [4 4];
block.OutputPort(1).DatatypeID  = 0; % double
block.OutputPort(1).Complexity  = 'Real';

% Register parameters
block.NumDialogPrms     = 0;

% Register sample times
%  [0 offset]            : Continuous sample time
%  [positive_num offset] : Discrete sample time
%
%  [-1, 0]               : Inherited sample time
%  [-2, 0]               : Variable sample time
block.SampleTimes = [-1 0];
```

```
%% ------------------------------------------------------------------
%% The M-file S-function uses an internal registry for all
%% block methods. You should register all relevant methods
%% (optional and required) as illustrated below. You may choose
%% any suitable name for the methods and implement these methods
%% as local functions within the same file. See comments
%% provided for each function for more information.
%% ------------------------------------------------------------------

block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
block.RegBlockMethod('InitializeConditions', @InitializeConditions);
block.RegBlockMethod('Start', @Start);
block.RegBlockMethod('Outputs', @Outputs);     % Required
block.RegBlockMethod('Update', @Update);
block.RegBlockMethod('Derivatives', @Derivatives);
block.RegBlockMethod('Terminate', @Terminate); % Required

%end setup

%%
%% PostPropagationSetup:
%%   Functionality    : Setup work areas and state variables. Can
%%                      also register run-time methods here
%%   Required         : No
%%   C-Mex counterpart: mdlSetWorkWidths
%%
function DoPostPropSetup(block)
block.NumDworks = 1;

  block.Dwork(1).Name            = 'x1';
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;      % double
  block.Dwork(1).Complexity      = 'Real'; % real
  block.Dwork(1).UsedAsDiscState = true;


%%
%% InitializeConditions:
%%   Functionality    : Called at the start of simulation and if it is
%%                      present in an enabled subsystem configured to reset
%%                      states, it will be called when the enabled subsystem
%%                      restarts execution to reset the states.
%%   Required         : No
%%   C-MEX counterpart: mdlInitializeConditions
%%
function InitializeConditions(block)

%end InitializeConditions


%%
%% Start:
%%   Functionality    : Called once at start of model execution. If you
%%                      have states that should be initialized once, this
%%                      is the place to do it.
%%   Required         : No
%%   C-MEX counterpart: mdlStart
%%
function Start(block)

block.Dwork(1).Data = 0;

%endfunction

%%
%% Outputs:
```

```
%%   Functionality    : Called to generate block outputs in
%%                      simulation step
%%   Required         : Yes
%%   C-MEX counterpart: mdlOutputs
%%
function Outputs(block)
% **** Custom Code by Loren Yeo (Start) ****

% **** Aide-Memoire ****

% For a linear system of the form
% dx = Ax
% or x_dot = Ax

% Consider a quadratic Lyapunov function candidate
% V = x'Px
% where P is a symmetric positive definite matrix

% Taking the derivative
% dV = dx'Px + x'Pdx = -x'Qx
% where A'P + PA = -Q

% For the system to be strictly stable, P must be SPD if Q is SPD

% **********************

% Position gain (lamba^2)
Kp = block.InputPort(1).Data(1);

% Velocity gain (2 * lambda)
Kv = block.InputPort(1).Data(2);

% Hurwitz gain matrix
A = [zeros(2) eye(2); -Kp*eye(2) -Kv*eye(2)];

% For computational simplicity, let Q = I
Q = eye(4);

% Output
P = lyap(A', Q);

block.OutputPort(1).Data = P;

% **** Custom Code by Loren Yeo (End) ****

%end Outputs

%%
%% Update:
%%   Functionality    : Called to update discrete states
%%                      during simulation step
%%   Required         : No
%%   C-MEX counterpart: mdlUpdate
%%
function Update(block)

% block.Dwork(1).Data = block.InputPort(1).Data;

%end Update

%%
%% Derivatives:
%%   Functionality    : Called to update derivatives of
%%                      continuous states during simulation step
%%   Required         : No
%%   C-MEX counterpart: mdlDerivatives
```

```
%%
function Derivatives(block)

%end Derivatives

%%
%% Terminate:
%%   Functionality    : Called at the end of simulation for cleanup
%%   Required         : Yes
%%   C-MEX counterpart: mdlTerminate
%%
function Terminate(block)

%end Terminate
```

A.5 Embedded MATLAB Code for Pack function in System Interface Subsystem

Pack function block as seen in Fig. 7.15 of Section 7.4.2

```
function out = pack(in)

% First define the size of the packet
% using nullcopy to avoid generating unnecessary code
out = eml.nullcopy(zeros(6, 1, 'int16'));

% I2C master has a driver flaw that prevents received unsigned integer
% value from exceeding 127 (maximum 7-bit unsigned integer value is 127)
% Workaround: Send 7 bits at a time instead of 8 bits
% Buffer is limited to a combined 14-bit maxiumum unsigned value of 16383
% Maximum buffer value is ascertained to not exceed 16383
% Therefore, two highest-order bits are unused and can be discarded

%{
% Sensor 1
sensor1xa = typecast(in(1), 'uint8');  % Lower 7-bit byte of X-coordinate
sensor1xb = typecast(in(2), 'uint8');  % Upper 7-bit byte of X-coordinate
sensor1ya = typecast(in(3), 'uint8');  % Lower 7-bit byte of Y-coordinate
sensor1yb = typecast(in(4), 'uint8');  % Upper 7-bit byte of Y-coordinate

% Sensor 2
sensor2xa = typecast(in(5), 'uint8');  % Lower 7-bit byte of X-coordinate
sensor2xb = typecast(in(6), 'uint8');  % Upper 7-bit byte of X-coordinate
sensor2ya = typecast(in(7), 'uint8');  % Lower 7-bit byte of Y-coordinate
sensor2yb = typecast(in(8), 'uint8');  % Upper 7-bit byte of Y-coordinate
%}

% Sensor 1
sensor1xa = in(1);      % Lower 7-bit byte of X-coordinate
sensor1xb = in(2);      % Upper 7-bit byte of X-coordinate
sensor1ya = in(3);      % Lower 7-bit byte of Y-coordinate
sensor1yb = in(4);      % Upper 7-bit byte of Y-coordinate
sensor1sqa = in(5);     % Lower 7-bit byte of surface quality
sensor1sqb = in(6);     % Upper 7-bit byte of surface quality

% Sensor 2
sensor2xa = in(7);      % Lower 7-bit byte of X-coordinate
sensor2xb = in(8);      % Upper 7-bit byte of X-coordinate
sensor2ya = in(9);      % Lower 7-bit byte of Y-coordinate
sensor2yb = in(10);     % Upper 7-bit byte of Y-coordinate
sensor2sqa = in(11);    % Lower 7-bit byte of surface quality
sensor2sqb = in(12);    % Upper 7-bit byte of surface quality

% Recast type for concatenation of upper and lower bytes
sensor1xa = cast(sensor1xa, 'uint16');
sensor1xb = cast(sensor1xb, 'uint16');
sensor1ya = cast(sensor1ya, 'uint16');
sensor1yb = cast(sensor1yb, 'uint16');
sensor1sqa = cast(sensor1sqa, 'uint16');
sensor1sqb = cast(sensor1sqb, 'uint16');
sensor2xa = cast(sensor2xa, 'uint16');
sensor2xb = cast(sensor2xb, 'uint16');
sensor2ya = cast(sensor2ya, 'uint16');
sensor2yb = cast(sensor2yb, 'uint16');
sensor2sqa = cast(sensor2sqa, 'uint16');
sensor2sqb = cast(sensor2sqb, 'uint16');

% Bit shift left logical for upper 7-bit bytes
sensor1xb = bitsll(sensor1xb, 7);
sensor1yb = bitsll(sensor1yb, 7);
```

```
sensor1sqb = bitsll(sensor1sqb, 7);
sensor2xb = bitsll(sensor2xb, 7);
sensor2yb = bitsll(sensor2yb, 7);
sensor2sqb = bitsll(sensor2sqb, 7);

% Bit-wise OR to combine upper and lower 7-bit bytes
sensor1x_uint14 = bitor(sensor1xa, sensor1xb);
sensor1y_uint14 = bitor(sensor1ya, sensor1yb);
sensor1sq_uint14 = bitor(sensor1sqa, sensor1sqb);
sensor2x_uint14 = bitor(sensor2xa, sensor2xb);
sensor2y_uint14 = bitor(sensor2ya, sensor2yb);
sensor2sq_uint14 = bitor(sensor2sqa, sensor2sqb);

% Convert to signed 16-bit integer
if sensor1x_uint14 >= 8192
    sensor1x = int16(sensor1x_uint14) - 16384;
else
    sensor1x = int16(sensor1x_uint14);
end

if sensor1y_uint14 >= 8192
    sensor1y = int16(sensor1y_uint14) - 16384;
else
    sensor1y = int16(sensor1y_uint14);
end

if sensor2x_uint14 >= 8192
    sensor2x = int16(sensor2x_uint14) - 16384;
else
    sensor2x = int16(sensor2x_uint14);
end

if sensor2y_uint14 >= 8192
    sensor2y = int16(sensor2y_uint14) - 16384;
else
    sensor2y = int16(sensor2y_uint14);
end

% Coordinate system of vehicle has axes swapped with respect to sensor
% Vehicle axes are 90 degrees clockwise with respect to sensor axes
sensor1x_veh = sensor1y;
sensor1y_veh = -sensor1x;
sensor1sq = sensor1sq_uint14;
sensor2x_veh = sensor2y;
sensor2y_veh = -sensor2x;
sensor2sq = sensor2sq_uint14;

out = [sensor1x_veh sensor1y_veh sensor1sq sensor2x_veh sensor2y_veh...
    sensor2sq]';
```

A.6 Embedded MATLAB Code for c1 function in Hybrid Geometric Localisation Subsystem

c1 function block as seen in Fig. 7.18 of Section 7.4.2.3

```
function dx1a = c1(r1, dx1, dxA, dyA, adj_1)
%#eml

r_crit = 6.8e10;

if abs(r1) > r_crit
    dx1a = (dxA * cos(adj_1)) - (dyA * sin(adj_1));
else
    dx1a = dx1;
end
```

A.7 Embedded MATLAB Code for c2 function in Hybrid Geometric Localisation Subsystem

c2 function block as seen in Fig. 7.18 of Section 7.4.2.3

```
function dy1a = c2(r1, dy1, dxA, dyA, adj_1)
%#eml

r_crit = 6.8e10;

if abs(r1) > r_crit
    dy1a = (dxA * sin(adj_1)) + (dyA * cos(adj_1));
else
    dy1a = dy1;
end
```

A.8 Embedded MATLAB Code for c3 function in Hybrid Geometric Localisation Subsystem

c3 function block as seen in Fig. 7.18 of Section 7.4.2.3

```
function dx2a = c3(r2, dx2, dxK, dyK, adj_2)
%#eml

r_crit = 6.8e10;

if abs(r2) > r_crit
    dx2a = (dxK * cos(adj_2)) - (dyK * sin(adj_2));
else
    dx2a = dx2;
end
```

A.9 Embedded MATLAB Code for c4 function in Hybrid Geometric Localisation Subsystem

c4 function block as seen in Fig. 7.18 of Section 7.4.2.3

```
function dy2a = c4(r2, dy2, dxK, dyK, adj_2)
%#eml
```

```
r_crit = 6.8e10;

if abs(r2) > r_crit
    dy2a = (dxK * sin(adj_2)) + (dyK * cos(adj_2));
else
    dy2a = dy2;
end
```

A.10 Embedded MATLAB Code for c5 function in Hybrid Geometric Localisation Subsystem

c5 function block as seen in Fig. 7.18 of Section 7.4.2.3

```
function dx2a = c5(r2, psi, dx2, dxK, dyK, adj_2)
%#eml

r_crit = 6.8e10;

if abs(r2) > r_crit
    dx2a = (dxK * cos(psi + adj_2)) - (dyK * sin(psi + adj_2));
else
    dx2a = dx2;
end
```

A.11 Embedded MATLAB Code for c6 function in Hybrid Geometric Localisation Subsystem

c6 function block as seen in Fig. 7.18 of Section 7.4.2.3

```
function dy2a = c6(r2, psi, dy2, dxK, dyK, adj_2)
%#eml

r_crit = 6.8e10;

if abs(r2) > r_crit
    dy2a = (dxK * sin(psi + adj_2)) + (dyK * cos(psi + adj_2));
else
    dy2a = dy2;
end
```

A.12 Embedded MATLAB Code for cpsi function in Hybrid Geometric Localisation Subsystem

cpsi function block as seen in Fig. 7.18 of Section 7.4.2.3

```
function dpsia = cpsi(dpsi)
%#eml

psi_min = 4e-4;

if abs(dpsi) < psi_min
    dpsia = 0;
else
    dpsia = dpsi;
end
```

Appendix B

B.1 Code for Atmel ATmega32 Microcontroller - MouseController.c

Microcontroller program pertaining to Sections 8.1.2.2 to 8.1.2.6

```
/**** Aide Memoire for Atmel AVR ATmega32 Microcontroller *****


** Bit-masking and bit-shifting in registers **
Set bit to high (1):
PORTB = PORTB | (1 << bit position);
PORTB |= (1 << bit position);

Reset bit to low (0):
PORTB = PORTB & ~(1 << bit position);
PORTB &= ~(1 << bit position);

Flip bit value:
PORTB = PORTB ^ (1 << bit position);
PORTB ^= (1 << bit position);

Multiple settings
0 and 0: PORTB &= ~((1<<PORTB0) | (1<<PORTB1));
0 and 1: PORTB = (PORTB & ~(1<<PORTB1)) | (1<<PORTB0);
1 and 0: PORTB = (PORTB & ~(1<<PORTB0)) | (1<<PORTB1);
1 and 1: PORTB |= (1<<PORTB0) | (1<<PORTB1);



** Port B (SPI) **

* Port B Register *
PINB (Input pins): Read-only. Register where data is read from pins
DDRB (Data direction): Set to high to configure pin as output and low to configure it as input
PORTB (Data): Register where data is written to pins

* Port B Pin Assignment *
PB7: SCK (SPI Bus Serial Clock)
PB6: MISO (SPI Bus Master Input/Slave Output)
PB5: MOSI (SPI Bus Master Output/Slave Input)
PB4: SS' (SPI Slave Select Input)
PB3: AIN1 (Analog Comparator Negative Input) / OC0 (Timer/Counter0 Output Compare Match
Output)
PB2: AIN0 (Analog Comparator Positive Input) / INT2 (External Interrupt 2 Input)
PB1: T1 (Timer/Counter1 External Counter Input)
PB0: T0 (Timer/Counter0 External Counter Input) / XCK (USART External Clock Input/Output)

* SPI Control Register (SPCR) *
Bit 7 - SPIE (SPI Interrupt Enable): Set to high for interrupt to be executed when a serial
transfer is completed
Bit 6 - SPE (SPI Enable): Set to high to enable SPI
Bit 5 - DORD (Data Order): Set to high to send LSB first and low to send MSB first
Bit 4 - MSTR (Master/Slave Select): Set to high to configure AVR as master and low to
configure it as slave
Bit 3 - CPOL (Clock Polarity): If set to high, SCK is high when idle; if set to low, SCK is
low when idle
Bit 2 - CPHA (Clock Phase): Set to high to sample data on trailing edge and low to sample data
on leading edge
Bits 1 and 0 - SPR1 and SPR0 (SPI Clock Rate Select): Set frequency of clock signal

SPR1; SPR0: SCK frequency
0; 0: fosc/4
0; 1: fosc/16
1; 0: fosc/64
1; 1: fosc/128
where fosc: oscillator frequency (3.6864 MHz)

CPOL; CPHA: Sampling point
```

0; 0: Leading (Rising) Edge
0; 1: Trailing (Falling) Edge
1; 0: Leading (Falling) Edge
1; 1: Trailing (Rising) Edge

* SPI Status Register (SPSR) *
Bit 7 - SPIF (SPI Interrupt Flag): Read-only. Set when data transfer is complete and cleared when interrupt is executed
Bit 6 - WCOL (Write Colision Flag): Read-only. Set if the SPDR register is written to during data transfer
Bit 0 - SPI2x (Double SPI Speed): Set to high to double SCK frequency when in master mode

* SPI Data Register (SPDR) *
Bits 7 to 0 - MSB to LSB: Writing initiates transmission; reading causes Receive Shift Register buffer to be read

** Port C (I2C/TWI) **

* TWI Bit Rate Register (TWBR) *
Bits 7 to 0 - TWBR7 to TWBR0 (TWI Bit Rate Register): Set SCL clock frequency along with TWPS in master mode

* TWI Control Register (TWCR) *
Bit 7 - TWINT (TWI Interrupt Flag): Set by hardware when transmission is complete; flag cleared by setting to high
Bit 6 - TWEA (TWI Enable Acknowledge Bit): Set to high to enable ACK pulse when slave address or data is received
Bit 5 - TWSTA (TWI START Condition Bit): Set to high for device to become a master
Bit 4 - TWSTO (TWI STOP Condition Bit): Set to high to generate STOP condition in master or for error recovery in slave
Bit 3 - TWWC (TWI Write Collision Flag): Read-only. Set when attempting to write to TWDR when TWINT is low
Bit 2 - TWEN T(WI Enable Bit): Set to high to enable TWI operation and activate TWI interface
Bit 1 - Res (Reserved Bit): Read-only. Always set to low
Bit 0 - TWIE (TWI Interrupt Enable): Set to high to enable interrupt request

* TWI Status Register (TWSR) *
Bits 7 to 3 - TWS7 to TWS3 (TWI Status): Read-only. Contain codes that reflect status of the TWI logic
Bit 2 - Res (Reserved Bit): Read-only. Always set to low
Bits 1 and 0 - TWPS1 and TWPS0 (TWI Prescaler Bits): Set SCL clock frequency along with TWBR in master mode

* TWI Data Register (TWDR) *
Bits 7 to 0 - TWD7 to TWD0 (TWI Data Register): Contain next byte to be transmitted or last byte received

* TWI (Slave) Address Register (TWAR) *
Bits 7 to 1 - TWA6 to TWA0 (TWI (Slave) Address Register): Contain 7-bit slave address
Bit 0 - TWGCE (TWI General Call Recognition Enable Bit): Set to high to enable recognition of a General Call

** Port D (USART) **

* USART Control and Status Register A (UCSRA) *
Bit 7 - RXC (USART Receive Complete): Read-only. Set when there is unread data in receive buffer and cleared when empty
Bit 6 - TXC (USART Transmit Complete): Set when transmit shift register and transmit buffer (UDR) are empty
Bit 5 - UDRE (USART Data Register Empty): Read-only. Set if transmit buffer (UDR) is empty and ready to be written
Bit 4 - FE (Frame Error): Read-only. Set when first stop bit of the next character in receive buffer is zero
Bit 3 - DOR (Data OverRun): Read-only. Set when the receive buffer is full (two characters)
Bit 2 - PE (Parity Error): Read-only. Set if next character in the receive buffer had a parity error when received
Bit 1 - U2X (Double the USART Transmission Speed): For asynchronous operation only

```
Bit 0 - MPCM (Multi-processor Communication Mode) : Set to high to ignore incoming frames
without address information


* USART Control and Status Register B (UCSRB) *
Bit 7 - RXCIE (RX Complete Interrupt Enable): Set to high to enable interrupt on the RXC Flag
Bit 6 - TXCIE (TX Complete Interrupt Enable): Set to high to enable interrupt on the TXC Flag
Bit 5 - UDRIE (USART Data Register Empty Interrupt Enable): Set to high to enable interrupt on
the UDRE Flag
Bit 4 - RXEN (Receiver Enable): Set to high to enable the USART receiver
Bit 3 - TXEN (Transmitter Enable): Set to high to enable the USART transmitter
Bit 2 - UCSZ2 (Character Size): Sets the number of data bits in a frame the receiver and
transmitter use
Bit 1 - RXB8 (Receive Data Bit 8): Read-only. RXB8 is the ninth data bit of the received
character
Bit 0 - TXB8 (Transmit Data Bit 8): TXB8 is the ninth data bit in the character to be
transmitted


* USART Control and Status Register C (UCSRC) *
Bit 7 - URSEL (Register Select): Set to high to access UCSRC and low to access UBRRH (USART
Baud Rate Register - High)
Bit 6 - UMSEL (USART Mode Select): Set to high for synchronous operation and low for
asynchronous operation
Bit 5 - UPM1 (Parity Mode): Enable and set type of parity generation and check
Bit 4 - UPM0 (Parity Mode): Enable and set type of parity generation and check
Bit 3 - USBS (Stop Bit Select): Set to high for two stop bits to be inserted by transmitter
and low for one stop bit
Bit 2 - UCSZ1 (Character Size): Sets the number of data bits in a frame the receiver and
transmitter use
Bit 1 - UCSZ0 (Character Size): Sets the number of data bits in a frame the receiver and
transmitter use
Bit 0 - UCPOL (Clock Polarity): Sets the relationship between output change, input sample and
synchronous clock (XCK)

UPM1; UPM0: Parity Mode
0; 0: Disabled
0; 1: Reserved
1; 0: Enabled, Even Parity
1; 1: Enabled, Odd Parity

UCSZ2; UCSZ1; UCSZ0: Character Size
0; 0; 0: 5-bit
0; 0; 1: 6-bit
0; 1; 0: 7-bit
0; 1; 1: 8-bit
1; 0; 0: Reserved
1; 0; 1: Reserved
1; 1; 0: Reserved
1; 1; 1: 9-bit

UCPOL: Transmitted Data Changed (Output of TxD Pin); Received Data Sampled (Input on RxD Pin)
0: Rising XCK Edge; Falling XCK Edge
1: Falling XCK Edge; Rising XCK Edge

**** End of Aide Memoire ****/


 // #include <io.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <avr/io.h>
#include <avr/common.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/sfr_defs.h>
#include <avr/iom32.h>
#include <avr/portpins.h>
#include <util/atomic.h>
#include <util/delay.h>
#include <util/twi.h>
```

```
#include "spi.h"
#include "twi_slave.h"
#include "conf/global.h"

#define fosc 3686400UL                                          // Crystal oscillator
frequency in Hz
// #define F_CPU 3686400UL
#define usart_baud_rate 115200                                  // USART baud rate in bits per
second
#define ubbr_value (((fosc / (usart_baud_rate * 16UL))) - 1)    // USART Baud Rate Register
value (UL: unsigned long int)

static int USART_putchar(unsigned char c, FILE *stream);  // Sends char/string to USART port
// static int USART_putchar(char c, FILE *stream);    // Sends char/string to USART port
static FILE usart_out = FDEV_SETUP_STREAM(USART_putchar, NULL, _FDEV_SETUP_WRITE);
                                                        // Set up a buffer as an stdio stream

/*
extern volatile unsigned int ISR_status;                        // ISR status flag
extern volatile unsigned int ISR_status_1;                      // ISR status flag counter
extern volatile unsigned int ISR_status_2;                      // ISR status flag counter
extern volatile unsigned int ISR_status_3;                      // ISR status flag counter
extern volatile unsigned int ISR_status_4;                      // ISR status flag counter
extern volatile unsigned int ISR_status_5;                      // ISR status flag counter
extern volatile unsigned int ISR_status_6;                      // ISR status flag counter
extern volatile unsigned int ISR_status_7;                      // ISR status flag counter
extern volatile unsigned int ISR_status_8;                      // ISR status flag counter
extern volatile unsigned int ISR_status_9;                      // ISR status flag counter
extern volatile unsigned int ISR_status_10;                     // ISR status flag counter
extern volatile unsigned int ISR_status_11;                     // ISR status flag counter
extern volatile unsigned int ISR_status_12;                     // ISR status flag counter
*/

extern volatile unsigned int TWI_buf_out[];

unsigned int TWI_slaveAddress = 0x01;                           // Own TWI slave address
unsigned int TWI_reg_addr = 0x44;            // Slave register address specified by TWI master
static int buffer_x_1 = 0;
static int buffer_y_1 = 0;
static int buffer_x_2 = 0;
static int buffer_y_2 = 0;
static int buffer_squal_1 = 0;
static int buffer_squal_2 = 0;
// static unsigned int counter = 1;


// Initialise USART port on microcontroller
void USART_init(void)
{
    UCSRB = ((1<<RXEN) | (1<<TXEN));                  // Enable USART receiver and transmitter
    UCSRC = ((1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0));   // Enable access to UCSRC and configure
                                                      // 8-bit character size
    UBRRL = ubbr_value;
    UBRRH = (ubbr_value>>8);
}

// Sends character/string to USART port
int USART_putchar(unsigned char text, FILE *stream)
// int USART_putchar(char text, FILE *stream)
{
    while ((UCSRA & (1<<UDRE)) == 0);
    UDR = text;
    return 0;
}

// Initialise SPI port in master mode on microcontroller
void SPI_master_init(void)
{
    // PORTB = 0x00;                      // Set all values to low on Port B for initialisation
    DDRB = ((1<<PB7) | (1<<PB5) | (1<<PB4) | (1<<PB3));   // Set SCK, MOSI, SS_1 and SS_2 as
                                                          // outputs on Port B
    // DDRB &= ~(1<<PB6);                                   // Set MISO as input on Port B
    PORTB = ((1<<PB4) | (1<<PB3));   // Set SS_1 and SS_2 to high when SPI is not transmitting
    SPCR = ((1<<SPE) | (1<<MSTR));                  // Enable SPI and configure master status
```

```
    SPCR |= (1<<CPOL);                                  // Set clock polarity of MOSI
    SPCR |= (1<<CPHA);                                  // Set clock phase of MOSI
    // SPCR |= (1<<SPR1);                               // Set clock rate in conjunction with SPR0
    SPCR |= (1<<SPR0);                                  // Set clock rate in conjunction with SPR1
    // SPSR |= (1<<SPI2X);                              // Set high to double clock rate
}

// Transmit data via SPI port
unsigned int SPI_transmit(unsigned int saddrdata)
{
    SPDR = saddrdata;
    while (!(SPSR & (1<<SPIF)));                        // Wait for transmission to complete
    return SPDR;
}

// Read data from SPI Receive Shift Register
unsigned int SPI_read(unsigned int saddr)
{
    // saddr &= ~(1<<7);                                // Set MSB to "0"
    unsigned int data1 = SPI_transmit(saddr);
    _delay_us(150);                                     // Set delay for 150 microseconds
    unsigned int data2 = SPI_transmit(0x00);
    _delay_us(20);                                      // Set delay for 20 microseconds
    return data2;
}

// Write data to SPI Transmit Shift Register
void SPI_write(unsigned int saddr, unsigned int sdata)
{
    saddr |= (1<<7);                                    // Set MSB to "1"
    unsigned int data1 = SPI_transmit(saddr);
    unsigned int data2 = SPI_transmit(sdata);
    _delay_us(100);                                     // Set delay for 100 microseconds
}

// Activate SPI port on Sensor 1
void SPI_1_activate(void)
{
    PORTB &= ~(1<<PB4);                                 // Set SS_1 to low on Port B
    _delay_us(50);                                      // Set delay for 50 microseconds
}

// Deactivate SPI port on Sensor 1
void SPI_1_deactivate(void)
{
    PORTB |= (1<<PB4);                                  // Set SS_1 to high on Port B
    _delay_us(50);                                      // Set delay for 50 microseconds
}

// Activate SPI port on Sensor 2
void SPI_2_activate(void)
{
    PORTB &= ~(1<<PB3);                                 // Set SS_2 to low on Port B
    _delay_us(50);                                      // Set delay for 50 microseconds
}

// Deactivate SPI port on Sensor 2
void SPI_2_deactivate(void)
{
    PORTB |= (1<<PB3);                                  // Set SS_2 to high on Port B
    _delay_us(50);                                      // Set delay for 50 microseconds
}

// Check for negative values and adjust accordingly (data range: -128 to 127)
int check_negative(unsigned int raw_data)
{
    int adjusted_data;
    if (raw_data >= 128)
    {
        adjusted_data = raw_data - 256;
        // adjusted_data = (reg_0x02 ^ 0xff) - 1;   // Binary conversion using two's
                                                    // complement method
    }
    else
```

```
        adjusted_data = raw_data;
    return adjusted_data;
}

// Convert decimal value into binary array
unsigned int *dec_to_bin(unsigned int dec_num)
{
    static unsigned int bits_0_to_7[8];
    memset(bits_0_to_7, 0, sizeof(bits_0_to_7));    // Reset array values to 0

    for (int i = 0; dec_num != 0; i++)
    {
        bits_0_to_7[i] = dec_num % 2;
        dec_num = dec_num / 2;
    }

    return bits_0_to_7;
}

// Print binary data in reverse order
void print_array(unsigned int *bin_array)
{
    for (int j = 7; j >= 0; j--)
    {
        printf("%u", bin_array[j]);                 // Print Bits 7 to 0 from left to right
    }
    printf("\n\n\r");
}

// Configure settings for optical sensor
void sensor_config(void)
{
    unsigned int reg_0x0a = SPI_read(0x0a);
    printf("Set resoulution to 800 counts per inch:\n\r");
    // reg_0x0a |= (1<<2);                              // Set Bit 2 to configure resolution
    reg_0x0a |= (1<<4);                                 // Set Bit 4 to configure resolution
    // reg_0x0a &= ~(1<<6);     // Set Bit 6 to low to turn shutter mode off (laser always on)
    SPI_write(0x0a, reg_0x0a);
    reg_0x0a = SPI_read(0x0a);
    printf("Configuration_bits (0x0a): 0x%x\n\r", reg_0x0a);
    printf("Configuration_bits (0x0a): 0b");
    unsigned int *reg_0x0a_bin;
    reg_0x0a_bin = dec_to_bin(reg_0x0a);
    print_array(reg_0x0a_bin);

    printf("Set relative laser current to 100%%:\n\r");
    SPI_write(0x2c, 0x00);              // Set Bits 0 to 6 to configure relative laser current
    unsigned int reg_0x2c = SPI_read(0x2c);
    printf("LP_CFG0 (0x2c): 0x%x\n\r", reg_0x2c);
    printf("LP_CFG0 (0x2c): 0b");
    unsigned int *reg_0x2c_bin;
    reg_0x2c_bin = dec_to_bin(reg_0x2c);
    print_array(reg_0x2c_bin);

    SPI_write(0x2d, 0xff);              // Set Bits 0 to 6 as complement of 0x2c register
    unsigned int reg_0x2d = SPI_read(0x2d);
    printf("LP_CFG1 (0x2d): 0x%x\n\r", reg_0x2d);
    printf("LP_CFG1 (0x2d): 0b");
    unsigned int *reg_0x2d_bin;
    reg_0x2d_bin = dec_to_bin(reg_0x2d);
    print_array(reg_0x2d_bin);

    SPI_write(0x12, 0xff);              // Clear motion registers Delta_X and Delta_Y
}

// Query sensor for delta x data
int read_delta_x(void)
{
    unsigned int reg_0x03 = SPI_read(0x03);
    int reg_0x03_adjusted = check_negative(reg_0x03);
    return reg_0x03_adjusted;
    // return reg_0x03;
}
```

```
// Query sensor for delta y data
int read_delta_y(void)
{
    unsigned int reg_0x04 = SPI_read(0x04);
    int reg_0x04_adjusted = check_negative(reg_0x04);
    return reg_0x04_adjusted;
    // return reg_0x04;
}

// Query sensor for surface quality data
unsigned int read_squal(void)
{
    unsigned int reg_0x05 = SPI_read(0x05);
    return reg_0x05;
}

// Convert signed 16-bit number to unsigned 14-bit (data range: 0 to 16383)
unsigned int convert_uint14(int data_int16)
{
    unsigned int data_uint14;
    if (data_int16 < 0)
    {
        data_uint14 = data_int16 + 16384;
    }
    else
        data_uint14 = data_int16;
    return data_uint14;
}

void TWI_buffer_load(void)
{
    // I2C master has driver flaw that prevents transmitted uint8 value from exceeding 127
    // Workaround: Send 7 bits at a time instead of 8 bits (maximum 7-bit uint8 value is 127)
    // Buffer does not exceed a 14-bit maxiumum unsigned value of 16383
    // Therefore, two highest-order bits are unused and can be discarded

    // Convert buffer value from signed 16-bit to unsigned 14-bit
    unsigned int buffer_x_1_uint14 = convert_uint14(buffer_x_1);
    unsigned int buffer_y_1_uint14 = convert_uint14(buffer_y_1);
    unsigned int buffer_x_2_uint14 = convert_uint14(buffer_x_2);
    unsigned int buffer_y_2_uint14 = convert_uint14(buffer_y_2);

    // Split 14-bit buffer value into two 7-bit chunks; eighth bit padded with zero
    TWI_buf_out[0] = (buffer_x_1_uint14 & 0x7F);
    TWI_buf_out[1] = (buffer_x_1_uint14 >> 7);
    TWI_buf_out[2] = (buffer_y_1_uint14 & 0x7F);
    TWI_buf_out[3] = (buffer_y_1_uint14 >> 7);
    TWI_buf_out[4] = (buffer_squal_1 & 0x7F);
    TWI_buf_out[5] = (buffer_squal_1 >> 7);

    TWI_buf_out[6] = (buffer_x_2_uint14 & 0x7F);
    TWI_buf_out[7] = (buffer_x_2_uint14 >> 7);
    TWI_buf_out[8] = (buffer_y_2_uint14 & 0x7F);
    TWI_buf_out[9] = (buffer_y_2_uint14 >> 7);
    TWI_buf_out[10] = (buffer_squal_2 & 0x7F);
    TWI_buf_out[11] = (buffer_squal_2 >> 7);

    // Reset cummulative buffer
    buffer_x_1 = 0;
    buffer_y_1 = 0;
    buffer_x_2 = 0;
    buffer_y_2 = 0;
    buffer_squal_1 = 0;
    buffer_squal_2 = 0;
    // counter = 1;
}


int main(void)
{
    // Initialise SPI port in master mode on microcontroller
    printf("Initialising SPI master... \n\r");
    SPI_master_init();
```

```
// Initialise TWI module for slave operation. Include address and/or enable General Call.
printf("Initialising TWI slave... \n\r");
TWI_Slave_Initialise((unsigned int)((TWI_slaveAddress<<TWI_ADR_BITS) | (1<<TWI_GEN_BIT)));

// Start the TWI transceiver to enable reception of the first command from the TWI Master
printf("Starting TWI slave transceiver... \n\r");
TWI_Start_Transceiver();

// Initialise USART port on microcontroller
USART_init();
stdout = &usart_out;

// Initialise both sensors (sensors share all pins with the exception of Slave Select)
DDRB |= (1<<PB1);              // Set as output on PORT B to provide RESET input on sensor
DDRB |= (1<<PB0);              // Set as output on PORT B to provide NPD input on sensor
PORTB |= (1<<PB0);             // Set NPD to high on Port B

printf("Resetting both Avago ADNS-6010 sensors...\n\r");
PORTB |= (1<<PB1);                              // Set RESET to high on Port B
_delay_ms(1);                                   // Set delay for 1 millisecond
PORTB &= ~(1<<PB1);                             // Set RESET to low on Port B
_delay_ms(200);                                 // Set delay for 200 milliseconds
printf("Sensor reset complete.\n\n\r");

// Configure settings for Sensor 1
printf("Configuring settings for Sensor 1...\n\n\r");
SPI_1_activate();
sensor_config();
SPI_1_deactivate();

// Configure settings for Sensor 2
printf("Configuring settings for Sensor 2...\n\n\r");
SPI_2_activate();
sensor_config();
SPI_2_deactivate();

// Enable I2C/TWI global interrupts
sei();

// Commence reading of sensors
printf("Reading coordinates...\n\n\r");


/* Note on sensors' motion detection:
The physical locations of the two sensors make it impossible for the vehicle to move
without both sensors registering motion. Hence, it is only necessary to query either
sensor for its motion status before reading data from both.
*/

// Poll motion register and read SPI data only when motion is detected
SPI_1_activate();

// unsigned int no_activity_count = 0;
// unsigned int ISR_status_12a = 1;
// unsigned int cycle = 0;

while (1)
{
    // Read motion register on Sensor 1; freeze coordinate data
    unsigned int reg_0x02 = SPI_read(0x02);
    unsigned int *reg_0x02_bin = dec_to_bin(reg_0x02);

    // if (no_activity_count > 12000)
    // {
    //      break;
    // }

    if (reg_0x02_bin[7] == 1)                       // Check whether motion is detected
    {
        int delta_x_1 = read_delta_x();
        int delta_y_1 = read_delta_y();
        unsigned int squal_1 = read_squal();
        // unsigned int delta_x_1_raw = read_delta_x();
        // int delta_x_1 = check_negative(delta_x_1_raw);
```

```
    // unsigned int delta_y_1_raw = read_delta_y();
    // int delta_y_1 = check_negative(delta_y_1_raw);

    SPI_1_deactivate();

    SPI_2_activate();

     // Read motion register on Sensor 2; freeze coordinate data
    reg_0x02 = SPI_read(0x02);
    int delta_x_2 = read_delta_x();
    int delta_y_2 = read_delta_y();
    unsigned int squal_2 = read_squal();
    // unsigned int delta_x_2_raw = read_delta_x();
    // int delta_x_2 = check_negative(delta_x_2_raw);
    // unsigned int delta_y_2_raw = read_delta_y();
    // int delta_y_2 = check_negative(delta_y_2_raw);

    SPI_2_deactivate();

    ATOMIC_BLOCK(ATOMIC_FORCEON)
    {
        buffer_x_1 += delta_x_1;
        buffer_y_1 += delta_y_1;
        buffer_squal_1 += squal_1;
        buffer_x_2 += delta_x_2;
        buffer_y_2 += delta_y_2;
        buffer_squal_2 += squal_2;
    }

    // counter++;

    // printf("          Delta X     Delta Y     SQual\n\r");
    // printf("Sensor 1: %7i %11i %9i\n\r", delta_x_1, delta_y_1, squal_1);
    // printf("Sensor 2: %7i %11i %9i\n\n\r", delta_x_2, delta_y_2, squal_2);

    // printf("          Delta X     Delta Y     Counts\n\r");
    // printf("Sensor 1: %7i %11i\n\r", delta_x_1, delta_y_1);
    // printf("Sensor 2: %7i %11i %10i\n\n\r", delta_x_2, delta_y_2, counter);

    // printf("%i\n", delta_x_1);
    // printf("%i\n", delta_y_1);
    // printf("%i\n", delta_x_2);
    // printf("%i\n", delta_y_2);

    SPI_1_activate();

    // no_activity_count = 0;
}
// no_activity_count++;

/*
if (ISR_status_12a > 150)
{
    printf("ISR_status_1: %i\n\r", ISR_status_1);
    printf("ISR_status_2: %i\n\r", ISR_status_2);
    printf("ISR_status_3: %i\n\r", ISR_status_3);
    printf("ISR_status_4: %i\n\r", ISR_status_4);
    printf("ISR_status_5: %i\n\r", ISR_status_5);
    printf("ISR_status_6: %i\n\r", ISR_status_6);
    printf("ISR_status_7: %i\n\r", ISR_status_7);
    printf("ISR_status_8: %i\n\r", ISR_status_8);
    printf("ISR_status_9: %i\n\r", ISR_status_9);
    printf("ISR_status_10: %i\n\r", ISR_status_10);
    printf("ISR_status_11: %i\n\r", ISR_status_11);
    printf("ISR_status_12: %i\n\r", ISR_status_12);
    printf("Cycle count: %i\n\r", cycle);
    exit(0);
}

if (ISR_status == 12)
{
    printf("ISR status: %ia  (%i)\n\r", ISR_status, ISR_status_12a);
    ISR_status_12a++;
}
```

```
        else
        {
            printf("ISR status: %i\n\r", ISR_status);
            ISR_status_12a = 1;
        }

        cycle++;
        */
    }
    // printf("\n\n\rProgram stopped due to lack of activity.");

/*
    // Constant data polling
    while (1)
    {
        SPI_1_activate();

        // Read motion register on Sensor 1; freeze coordinate data
        unsigned int reg_0x02 = SPI_read(0x02);
        int delta_x_1 = read_delta_x();
        int delta_y_1 = read_delta_y();
        // unsigned int delta_x_1_raw = read_delta_x();
        // int delta_x_1 = check_negative(delta_x_1_raw);
        // unsigned int delta_y_1_raw = read_delta_y();
        // int delta_y_1 = check_negative(delta_y_1_raw);

        SPI_1_deactivate();

        SPI_2_activate();

        // Read motion register on Sensor 1; freeze coordinate data
        reg_0x02 = SPI_read(0x02);
        int delta_x_2 = read_delta_x();
        int delta_y_2 = read_delta_y();
        // unsigned int delta_x_2_raw = read_delta_x();
        // int delta_x_2 = check_negative(delta_x_2_raw);
        // unsigned int delta_y_2_raw = read_delta_y();
        // int delta_y_2 = check_negative(delta_y_2_raw);

        SPI_2_deactivate();

        ATOMIC_BLOCK(ATOMIC_FORCEON)
        {
            buffer_x_1 += delta_x_1;
            buffer_y_1 += delta_y_1;
            buffer_x_2 += delta_x_2;
            buffer_y_2 += delta_y_2;
        }

        // printf("          Delta X     Delta Y\n\r");
        // printf("Sensor 1: %7i %11i\n\r", delta_x_1, delta_y_1);
        // printf("Sensor 2: %7i %11i\n\n\r", delta_x_2, delta_y_2);

        // printf("%i\n", delta_x_1);
        // printf("%i\n", delta_y_1);
        // printf("%i\n", delta_x_2);
        // printf("%i\n", delta_y_2);
    }
*/

    SPI_1_deactivate();
    SPI_2_deactivate();

}
```

B.2 Code for Atmel ATmega32 Microcontroller - twi_slave.c

I^2C/TWI source file for microcontroller program pertaining to Section 8.1.2.5

Note: Code supplied by Atmel, edited by Bernhard Walle for GNU C compatibility, and modified by Loren Yeo for the project

```
/****************************************************************************
*
* Atmel Corporation
*
* File              : TWI_Slave.c
* Compiler          : IAR EWAAVR 2.28a/3.10c
* Revision          : $Revision: 1.7 & 2475 $
* Date              : $Date: Thursday, Aug 05, 2004 09:22:50 UTC & 2007-09-20 12:00:43 +0200 $
* Updated by        : $Author: lholsen & mlarsson $
*
* Support mail      : avr@atmel.com
*
* Supported devices : All devices with a TWI module can be used.
*                     The example is written for the ATmega16
*
* AppNote           : AVR311 - TWI Slave Implementation
*
* Description       : This is sample driver to AVRs TWI module.
*                     It is interupt driven. All functionality is controlled through
*                     passing information to and from functions. See main.c for samples
*                     of how to use the driver.
*
* Note              : Revision 1.7 edited by Bernhard Walle to be compatible with the GNU C
*                     Compiler and AVR-libc.
*                     Additional changes to both Revisions 1.7 and 2475 made by Loren Yeo.
*
****************************************************************************/


#include <avr/io.h>
#include <stdbool.h>
#include <stdio.h>

#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "global.h"
#include "twi_slave.h"

static unsigned int TWI_buf[TWI_BUFFER_SIZE];       // Transceiver buffer (outgoing)
static unsigned int TWI_msgSize = 0;                // Number of bytes to be transmitted
static unsigned int TWI_state = TWI_NO_STATE;       // State byte; default set to TWI_NO_STATE
enum {TWI_data_size = 12};                          // Size of one set of data in bytes
volatile unsigned int TWI_buf_in;                   // Transceiver buffer (incoming)
volatile unsigned int TWI_buf_out[TWI_data_size];   // Transceiver buffer (outgoing)
extern unsigned int TWI_reg_addr;                   // Register addr specified by TWI master

/*
volatile unsigned int ISR_status = 0;               // ISR status flag
volatile unsigned int ISR_status_1 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_2 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_2a = 0;            // ISR status flag counter
volatile unsigned int ISR_status_3 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_4 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_5 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_6 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_7 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_8 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_9 = 0;             // ISR status flag counter
volatile unsigned int ISR_status_10 = 0;            // ISR status flag counter
volatile unsigned int ISR_status_11 = 0;            // ISR status flag counter
volatile unsigned int ISR_status_12 = 0;            // ISR status flag counter
*/
```

```
/*
This is true when the TWI is in the middle of a transfer
and set to false when all bytes have been transmitted/received
Also used to determine how deep we can sleep.
*/
// static unsigned char TWI_busy = 0;              // (Rev 2475)


union TWI_statusReg_t TWI_statusReg = {0};    // TWI_statusReg defined in TWI_Slave.h (R 2475)
// union TWI_statusReg TWI_statusReg = {0};   // TWI_statusReg defined in TWI_Slave.h (R 1.7)


/****************************************************************************
Call this function to set up the TWI slave to its initial standby state.
Remember to enable interrupts from the main application after initializing the TWI.
Pass both the slave address and the requrements for triggering on a general call in the
same byte. Use e.g. this notation when calling this function:
TWI_Slave_Initialise( (TWI_slaveAddress<<TWI_ADR_BITS) | (TRUE<<TWI_GEN_BIT) );
The TWI module is configured to NACK on any requests. Use a TWI_Start_Transceiver function to
start the TWI.
****************************************************************************/
void TWI_Slave_Initialise(unsigned int TWI_ownAddress)
{
    TWAR = TWI_ownAddress;              // Set own TWI slave address. Accept TWI General Calls
    // TWDR = 0xFF;                     // Default content = SDA released (Rev 1.7)
    TWCR = (1<<TWEN)|                   // Enable TWI-interface and release TWI pins
           (0<<TWIE)|(0<<TWINT)|                    // Disable TWI Interupt
           (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|         // Do not ACK on any requests, yet
           (0<<TWWC);
    // TWI_busy = 0;                                // (Rev 2475)
}


/****************************************************************************
Call this function to test if the TWI_ISR is busy transmitting.
****************************************************************************/
unsigned int TWI_Transceiver_Busy(void)
{
    // If TWI interrupt is enabled then the Transceiver is busy
    // return (TWCR & (1<<TWIE));                   // (Rev 1.7)
    // return TWI_busy;                             // (Rev 2475)
    return (TWCR & (1<<TWINT));
}


/****************************************************************************
Call this function to fetch the state information of the previous operation. The function will
hold execution (loop) until the TWI_ISR has completed with the previous operation. If there
was
an error, then the function will return the TWI State code.
****************************************************************************/
unsigned int TWI_Get_State_Info(void)
{
    while (TWI_Transceiver_Busy());             // Wait until TWI has completed transmission
    return (TWI_state);                         // Return error state
}


/****************************************************************************
Call this function to send a prepared message, or start the Transceiver for reception. Include
a pointer to the data to be sent if a SLA+W is received. The data will be copied to the TWI
buffer. Also include how many bytes that should be sent. Note that unlike the similar Master
function, the Address byte is not included in the message buffers.
The function will hold execution (loop) until the TWI_ISR has completed with the previous
operation,
then initialize the next operation and return.
****************************************************************************/
void TWI_Start_Transceiver_With_Data(unsigned int *msg, unsigned int msgSize)
{
    unsigned int temp;

    while (TWI_Transceiver_Busy());         // Wait until TWI is ready for next transmission
    TWI_msgSize = msgSize;                  // Number of data to transmit
```

```
    for (temp = 0; temp < msgSize; temp++)    // Copy data that may be transmitted if the TWI
                                              //  Master requests data
    {
        // TWI_buf[temp] = msg[temp];
        TWI_buf_out[temp] = msg[temp];
    }
    TWI_statusReg.all = 0;
    TWI_state = TWI_NO_STATE;
    TWCR = (1<<TWEN)|                         // TWI Interface enabled
           (1<<TWIE)|(1<<TWINT)|              // Enable TWI Interupt and clear the flag
           (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|   // Prepare to ACK next time Slave is addressed
           (0<<TWWC);
    // TWI_busy = 1;                              // (Rev 2475)
}


/****************************************************************************
Call this function to fill buffer with data to be transmitted.
****************************************************************************/
void TWI_load_buffer(unsigned int *msg, unsigned int msgSize)
{
    while (TWI_Transceiver_Busy());           // Wait until TWI is ready for next transmission
    TWI_msgSize = msgSize;                    // Number of data to transmit
    for (int idx = 0; idx < msgSize; idx++)   // Copy data that may be transmitted if the TWI
                                              //  Master requests data
    {
        TWI_buf_out[idx] = msg[idx];
    }
    // TWCR |= (1<<TWINT);
}


/****************************************************************************
Call this function to start the Transceiver without specifing new transmission data.
Useful for restarting a transmission, or just starting the transceiver for reception.
The driver will reuse the data previously put in the transceiver buffers. The function will
hold execution (loop) until the TWI_ISR has completed with the  previous operation, then
initialize the next operation and return.
****************************************************************************/
void TWI_Start_Transceiver(void)
{
    while (TWI_Transceiver_Busy());           // Wait until TWI is ready for next transmission
    TWI_statusReg.all = 0;
    TWI_state = TWI_NO_STATE;
    TWCR = (1<<TWEN)|                         // TWI Interface enabled
           (1<<TWIE)|(1<<TWINT)|              // Enable TWI Interupt and clear the flag
           (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|   // Prepare to ACK next time Slave is addressed
           (0<<TWWC);
    // TWI_busy = 0;                              // (Rev 2475)
}


/****************************************************************************
Call this function to read out the received data from the TWI transceiver buffer. I.e. first
call TWI_Start_Transceiver to get the TWI Transceiver to fetch data. Then Run this function to
collect the data when they have arrived. Include a pointer to where to place the data and
the number of bytes to fetch in the function call. The function will hold execution (loop)
until the TWI_ISR has completed with the previous operation, before reading out the data
and returning. If there was an error in the previous transmission the function will return
the TWI State code.
****************************************************************************/
unsigned int TWI_Get_Data_From_Transceiver(unsigned int *msg, unsigned int msgSize)
{
    unsigned int i;
    while (TWI_Transceiver_Busy());           // Wait until TWI is ready for next transmission
    if(TWI_statusReg.slv.lastTransOK)         // Last transmission completed successfully
    {
        for (i = 0; i < msgSize; i++)         // Copy data from Transceiver buffer
        {
            msg[i] = TWI_buf[i];
            // printf("Data %d: %d\n", i, msg[i]);
        }
        TWI_statusReg.slv.RxDataInBuf = FALSE; // Slave Receive data has been read from buffer
    }
```

```
    return TWI_statusReg.slv.lastTransOK;
}


/****************************************************************************
This function is the Interrupt Service Routine (ISR), and called when the TWI interrupt is
triggered; that is whenever a TWI event has occurred. This function should not be called
directly from the main application.
****************************************************************************/
// SIGNAL(SIG_2WIRE_SERIAL)
ISR(TWI_vect)
{
    unsigned int TWSR_status = (TWSR & TWSR_STATUS_MASK);
    static unsigned int TWI_bufPtr;

    switch (TWSR_status)
    {
        // Own SLA+R has been received; ACK has been returned
        case TWI_STX_ADR_ACK:
            // ISR_status = 1;
            // ISR_status_1++;

        // Arbitration lost in SLA+R/W as Master; own SLA+R has been received; ACK returned
        // case TWI_STX_ADR_ACK_M_ARB_LOST:
            TWI_bufPtr = 0;                     // Set buffer pointer to first data location
            TWI_buffer_load();

        // Data byte in TWDR has been transmitted; ACK has been received
        case TWI_STX_DATA_ACK:
            // ISR_status = 2;
            // ISR_status_2++;
            /*
            if ((TWI_buf_in == TWI_reg_addr) && (TWI_bufPtr < TWI_msgSize))
            {
                TWDR = TWI_buf_out[TWI_bufPtr++];
            }
            */
            TWDR = TWI_buf_out[TWI_bufPtr++];
            TWCR = (1<<TWEN)|                               // TWI Interface enabled
                    (1<<TWIE)|(1<<TWINT)|  // Enable TWI Interrupt and clear flag to send byte
                    (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|
                    (0<<TWWC);
            // ISR_status_2a++;
            // TWI_busy = 1;                            // (Rev 2475)
            break;

        // Data byte in TWDR has been transmitted; NACK has been received.
        // I.e. this could be the end of the transmission.
        case TWI_STX_DATA_NACK:
            // ISR_status = 3;
            // ISR_status_3++;
            /*
            if (TWI_bufPtr == TWI_msgSize)          // Have we transceived all expected data?
            {
                TWI_statusReg.slv.lastTransOK = TRUE;   // Set status bits to completed
            }
            else                            // Master has sent a NACK before all data were sent
            {
                TWI_state = TWSR_status;                // Store TWI State as error message
            }
            */
            TWCR = (1<<TWEN)|                   // Enable TWI-interface and release TWI pins
                    (1<<TWIE)|(1<<TWINT)|// Keep interrupt enabled and clear the flag (R 2475)
                    (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|  // Answer on next address match (R 2475)
                    (0<<TWWC);
            /*
            // Put TWI Transceiver in passive mode (Rev 1.7)
            TWCR = (1<<TWEN)|                   // Enable TWI-interface and release TWI pins
                    (0<<TWIE)|(0<<TWINT)|               // Disable Interrupt (Rev 1.7)
                    (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|    // Do not acknowledge on any new
                                                        // requests (Rev 1.7)
                    (0<<TWWC);
            */
            // TWI_busy = 0;                // Transmit is finished; not busy anymore (Rev 2475)
```

```
    break;

// General call address has been received; ACK has been returned
case TWI_SRX_GEN_ACK:
    // ISR_status = 4;
    // ISR_status_4++;

// Arbitration lost in SLA+R/W as Master; General call address received; ACK returned
// case TWI_SRX_GEN_ACK_M_ARB_LOST:
    // TWI_statusReg.slv.genAddressCall = TRUE;

// Own SLA+W has been received ACK has been returned
case TWI_SRX_ADR_ACK:
    // ISR_status = 5;
    // ISR_status_5++;

// Arbitration lost in SLA+R/W as Master; own SLA+W has been received; ACK returned
// case TWI_SRX_ADR_ACK_M_ARB_LOST:
    // Dont need to clear TWI_S_statusReg.genAddressCall due to it being default state
    // TWI_statusReg.slv.RxDataInBuf = TRUE;
    TWI_bufPtr = 0;                         // Set buffer pointer to first data location
    // Reset the TWI Interupt to wait for a new event
    TWCR = (1<<TWEN)|                                // TWI Interface enabled
           (1<<TWIE)|(1<<TWINT)|  // Enable TWI Interrupt and clear flag to send byte
           (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|    // Expect ACK on this transmission
           (0<<TWWC);
    // TWI_busy = 1;                            // (Rev 2475)
    break;

// Previously addressed with own SLA+W; data has been received; ACK has been returned
case TWI_SRX_ADR_DATA_ACK:
    // ISR_status = 6;
    // ISR_status_6++;

// Previously addressed with general call; data has been received; ACK returned
case TWI_SRX_GEN_DATA_ACK:
    // ISR_status = 7;
    // ISR_status_7++;
    // TWI_buf[TWI_bufPtr++] = TWDR;
    TWI_buf_in = TWDR;
    // printf("Received data: %i\n\r", TWI_buf[TWI_bufPtr - 1]);
    // TWI_statusReg.slv.lastTransOK = TRUE;       // Set flag transmission successful
    // Reset the TWI Interupt to wait for a new event.
    TWCR = (1<<TWEN)|                                // TWI Interface enabled
           (1<<TWIE)|(1<<TWINT)|  // Enable TWI Interrupt and clear flag to send byte
           (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|    // Send ACK after next reception
           (0<<TWWC);
    // TWI_busy = 1;                            // (Rev 2475)
    break;

// A STOP or repeated START condition has been received while still addressed as Slave
case TWI_SRX_STOP_RESTART:
    // ISR_status = 8;
    // ISR_status_8++;
    // Enter not addressed mode and listen to address match (Rev 2475)
    TWCR = (1<<TWEN)|                       // Enable TWI-interface and release TWI pins
           (1<<TWIE)|(1<<TWINT)|     // Enable interrupt and clear the flag  (R 2475)
           (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|   // Wait for new address match  (R 2475)
           (0<<TWWC);
    /*
    // Put TWI Transceiver in passive mode (Rev 1.7)
    TWCR = (1<<TWEN)|                       // Enable TWI-interface and release TWI pins
           (0<<TWIE)|(0<<TWINT)|                // Disable Interrupt (Rev 1.7)
           (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|    // Do not acknowledge on any new
                                               // requests (Rev 1.7)
           (0<<TWWC);
    */
    // TWI_busy = 0;     // Waiting for a new address match, so is not busy (Rev 2475)
    break;

// Previously addressed with own SLA+W; data has been received; NOT ACK returned
case TWI_SRX_ADR_DATA_NACK:
    // ISR_status = 9;
    // ISR_status_9++;
```

```
        // Previously addressed with general call; data has been received; NOT ACK returned
        case TWI_SRX_GEN_DATA_NACK:
            // ISR_status = 10;
            // ISR_status_10++;

        // Last data byte in TWDR has been transmitted (TWEA = 0); ACK has been received
        case TWI_STX_DATA_ACK_LAST_BYTE:
            // ISR_status = 11;
            // ISR_status_11++;

        // No relevant state information available; TWINT = 0
        // case TWI_NO_STATE:

        // Bus error due to an illegal START or STOP condition
        case TWI_BUS_ERROR:
            // ISR_status = 12;
            // ISR_status_12++;
            TWI_state = TWSR_status;                    // Store TWI State as error message,
                                                        // operation also clears noErrors bit
            // Recover from TWI_BUS_ERROR
            TWCR |= ((1<<TWSTO)|(1<<TWINT));          // Release the SDA and SCL pins thus
                                                      // enabling other devices to use the bus
            break;

        default:
            // ISR_status = 13;
            // TWI_state = TWSR_status;               // Store TWI State as error message,
                                                      // operation also clears the Success bit
            TWCR = (1<<TWEN)|                       // Enable TWI-interface and release TWI pins
                   (1<<TWIE)|(1<<TWINT)|    // Keep interrupt enabled and clear flag (R 2475)
                   (1<<TWEA)|(0<<TWSTA)|(1<<TWSTO)|   // Acknowledge on new requests (R 2475)
                   (0<<TWWC);
            /*
            TWCR = (1<<TWEN)|                       // Enable TWI-interface and release TWI pins
                   (0<<TWIE)|(0<<TWINT)|               // Disable Interrupt (Rev 1.7)
                   (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|    // Do not acknowledge on any new
                                                       // requests (Rev 1.7)
                   (0<<TWWC);
            */
            // TWI_busy = 0;                          // Unknown status; wait for new address
                                                      //  match that can be handled (Rev 2475)

    }
}
```

B.3 Code for Atmel ATmega32 Microcontroller - twi_slave.h

I^2C/TWI header file for microcontroller program pertaining to Section 8.1.2.5

Note: Code supplied by Atmel, edited by Bernhard Walle for GNU C compatibility, and modified by Loren Yeo for the project

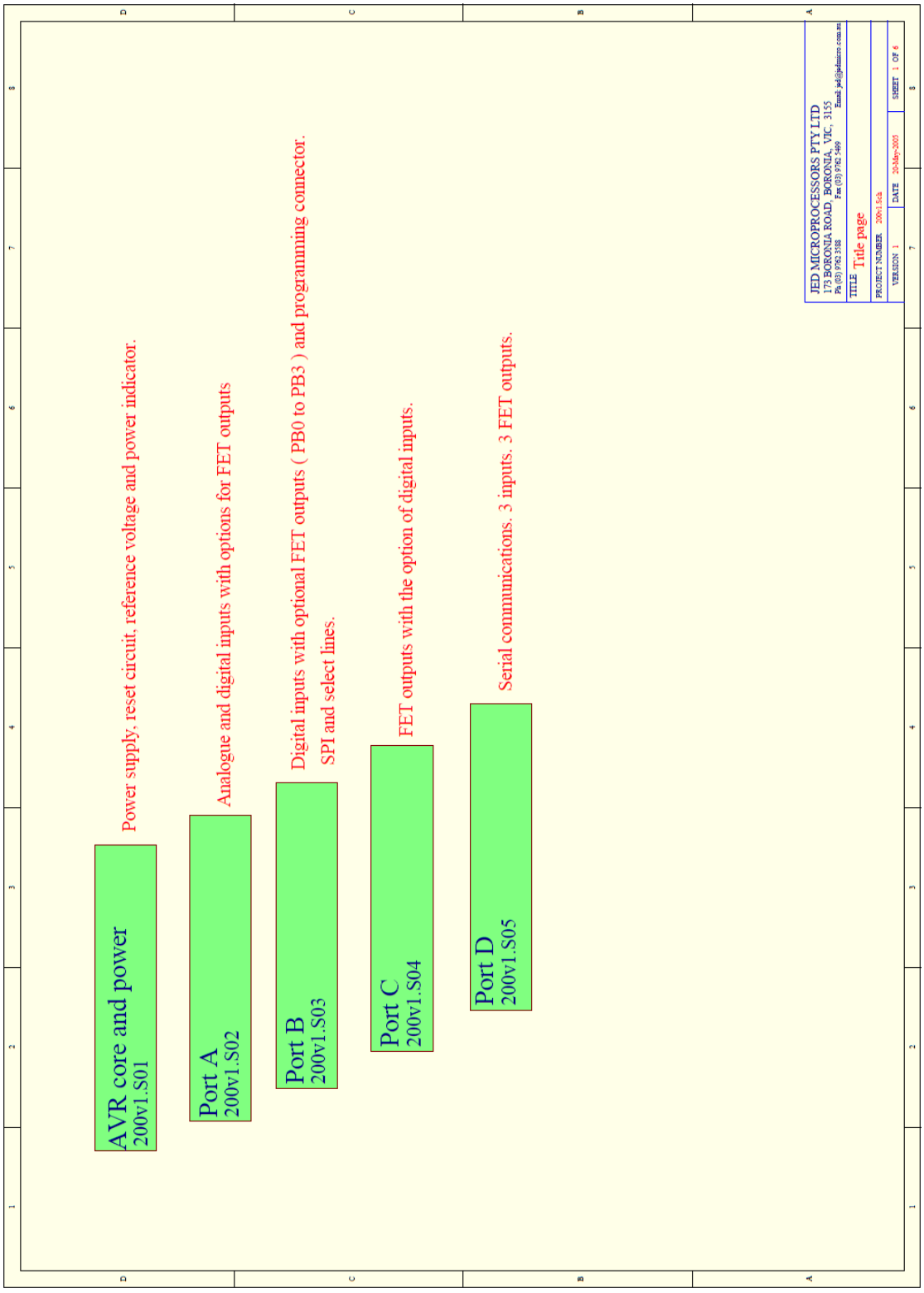
```
/****************************************************************************
*
* Atmel Corporation
*
* File              : TWI_Slave.h
* Compiler          : IAR EWAAVR 2.28a/3.10c
* Revision          : $Revision: 1.6 & 2475 $
* Date              : $Date: Monday, May 24, 2004 09:32:18 UTC & 2007-09-20 12:00:43 +0200 $
* Updated by        : $Author: ltwa & mlarsson $
*
* Support mail      : avr@atmel.com
*
* Supported devices : All devices with a TWI module can be used.
*                     The example is written for the ATmega16
*
* AppNote           : AVR311 - TWI Slave Implementation
*
* Description       : Header file for TWI_slave.c
*                     Include this file in the application.
*
* Note              : Revision 1.6 edited by Bernhard Walle to be compatible with the GNU C
*                     Compiler and AVR-libc.
*                     Additional changes to both Revisions 1.6 and 2475 made by Loren Yeo.
*
****************************************************************************/


#ifndef TWI_SLAVE_H
#define TWI_SLAVE_H

/****************************************************************************
  TWI Status/Control register definitions
****************************************************************************/

#define TWI_BUFFER_SIZE 6       // Reserves memory for the drivers transceiver buffer.
                                // Set this to the largest message size that will be sent
including
                                // address byte.

/****************************************************************************
  Global definitions
****************************************************************************/

union TWI_statusReg_t                           // Status byte holding flags (Revision 2475)
// union TWI_statusReg                          // Status byte holding flags (Revision 1.6)
{
    unsigned int all;
    struct
    {
        unsigned int lastTransOK:1;
        unsigned int RxDataInBuf:1;
        unsigned int genAddressCall:1;          // TRUE = General call, FALSE = TWI Address;
        unsigned int unusedBits:5;
    } slv;
};

extern union TWI_statusReg_t TWI_statusReg;     // (Revision 2475)
// extern union TWI_statusReg TWI_statusReg;    // (Revision 1.6)

/****************************************************************************
  Function definitions
****************************************************************************/
void TWI_Slave_Initialise(unsigned int);
unsigned int TWI_Transceiver_Busy(void);
```

```
unsigned int TWI_Get_State_Info(void);
void TWI_Start_Transceiver_With_Data(unsigned int*, unsigned int);
void TWI_Start_Transceiver(void);
unsigned int TWI_Get_Data_From_Transceiver(unsigned int*, unsigned int);
void TWI_load_buffer(unsigned int*, unsigned int);
void TWI_buffer_load(void);

/****************************************************************************
  Bit and byte definitions
****************************************************************************/
#define TWI_READ_BIT    0       // Bit position for R/W bit in "address byte"
#define TWI_ADR_BITS    1       // Bit position for LSB of the slave address bits in init byte
#define TWI_GEN_BIT     0       // Bit position for LSB of the general call bit in init byte

/****************************************************************************
  TWI State codes
****************************************************************************/
// General TWI Master status codes
#define TWI_START                  0x08    // START has been transmitted
#define TWI_REP_START              0x10    // Repeated START has been transmitted
#define TWI_ARB_LOST               0x38    // Arbitration lost

// TWI Master Transmitter status codes
#define TWI_MTX_ADR_ACK            0x18    // SLA+W has been transmitted and ACK received
#define TWI_MTX_ADR_NACK           0x20    // SLA+W has been transmitted and NACK received
#define TWI_MTX_DATA_ACK           0x28    // Data byte has been transmitted and ACK received
#define TWI_MTX_DATA_NACK          0x30   // Data byte has been transmitted and NACK received

// TWI Master Receiver status codes
#define TWI_MRX_ADR_ACK            0x40    // SLA+R has been transmitted and ACK received
#define TWI_MRX_ADR_NACK           0x48    // SLA+R has been transmitted and NACK received
#define TWI_MRX_DATA_ACK           0x50    // Data byte has been received and ACK transmitted
#define TWI_MRX_DATA_NACK          0x58   // Data byte has been received and NACK transmitted

// TWI Slave Transmitter status codes
#define TWI_STX_ADR_ACK            0xA8   // Own SLA+R has been received; ACK returned
#define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0    // Arbitration lost in SLA+R/W as Master; own
                                           // SLA+R has been received; ACK has been returned
#define TWI_STX_DATA_ACK           0xB8    // Data byte in TWDR has been transmitted; ACK
                                           // has been received
#define TWI_STX_DATA_NACK          0xC0    // Data byte in TWDR has been transmitted; NOT ACK
                                           // has been received
#define TWI_STX_DATA_ACK_LAST_BYTE 0xC8    // Last data byte in TWDR has been transmitted
                                           // (TWEA = 0); ACK has been received

// TWI Slave Receiver status codes
#define TWI_SRX_ADR_ACK            0x60    // Own SLA+W has been received ACK returned
#define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68    // Arbitration lost in SLA+R/W as Master; own
                                           // SLA+W has been received; ACK has been returned
#define TWI_SRX_GEN_ACK            0x70    // General call address has been received; ACK has
                                           // been returned
#define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78    // Arbitration lost in SLA+R/W as Master; Gen call
                                           // address has been received; ACK returned
#define TWI_SRX_ADR_DATA_ACK       0x80    // Previously addressed with own SLA+W; data has
                                           // been received; ACK has been returned
#define TWI_SRX_ADR_DATA_NACK      0x88    // Previously addressed with own SLA+W; data has
                                           // been received; NOT ACK has been returned
#define TWI_SRX_GEN_DATA_ACK       0x90    // Previously addressed with general call; data
                                           // has been received; ACK has been returned
#define TWI_SRX_GEN_DATA_NACK      0x98    // Previously addressed with general call; data
                                           // has been received; NOT ACK has been returned
#define TWI_SRX_STOP_RESTART       0xA0    // A STOP or repeated START condition has been
                                           // received while still addressed as Slave

// TWI Miscellaneous status codes
#define TWI_NO_STATE               0xF8 // No relevant state information available; TWINT = 0
#define TWI_BUS_ERROR              0x00  // Bus error due to illegal START or STOP condition

// Defines and constants
#define TWCR_CMD_MASK              0x0F
#define TWSR_STATUS_MASK           0xF8

#endif /* TWI_SLAVE_H */
```

Appendix C

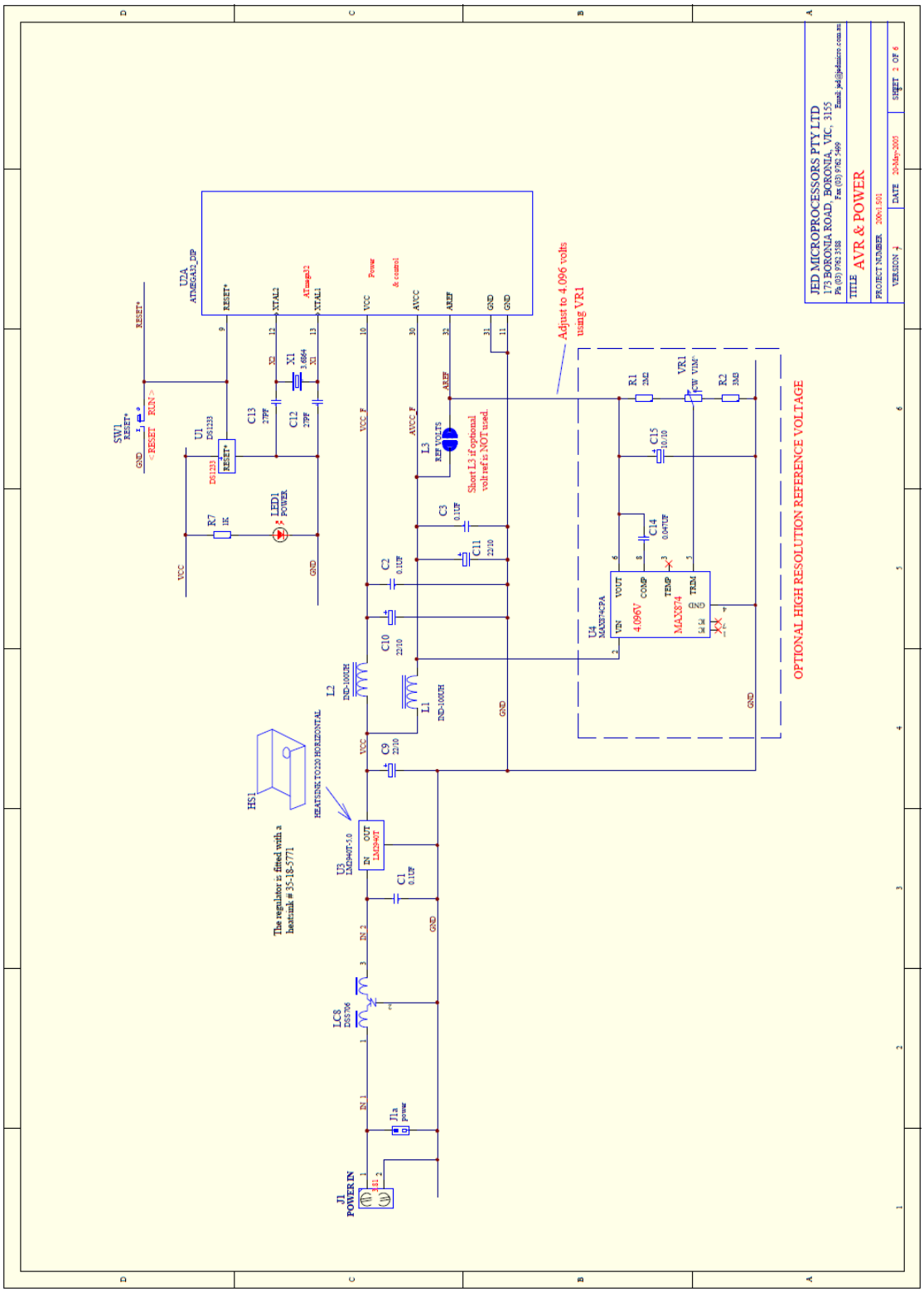
C.1 Circuit Schematics of JED AVR200 Single Board Computer

Circuit diagram pertaining to Section 8.1.2 (including Subsections 8.1.2.1 to 8.1.2.6)



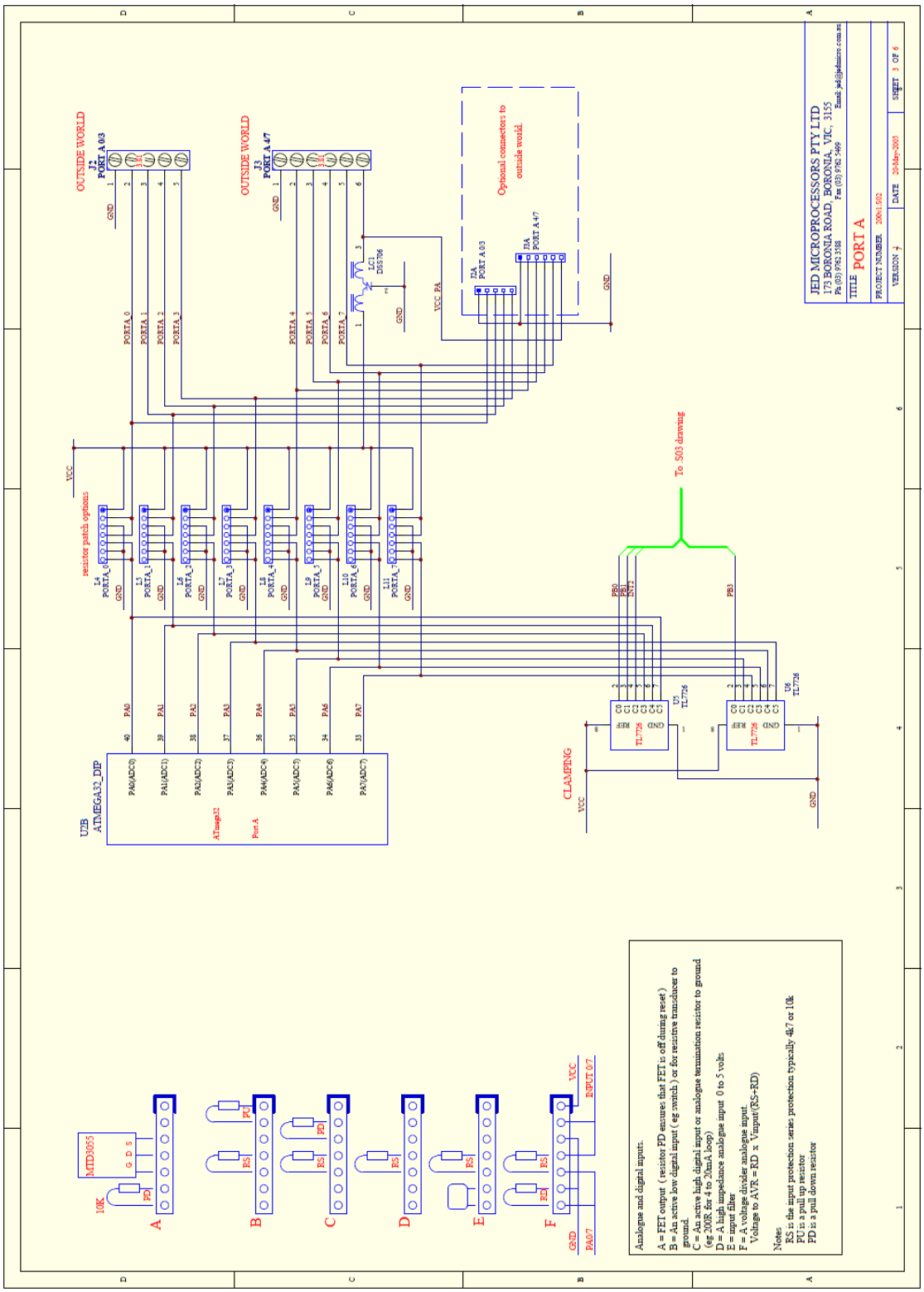
C.2 Circuit Schematics of Core and Power of JED AVR200

Circuit diagram pertaining to Section 8.1.2 (including Subsections 8.1.2.1 to 8.1.2.6)



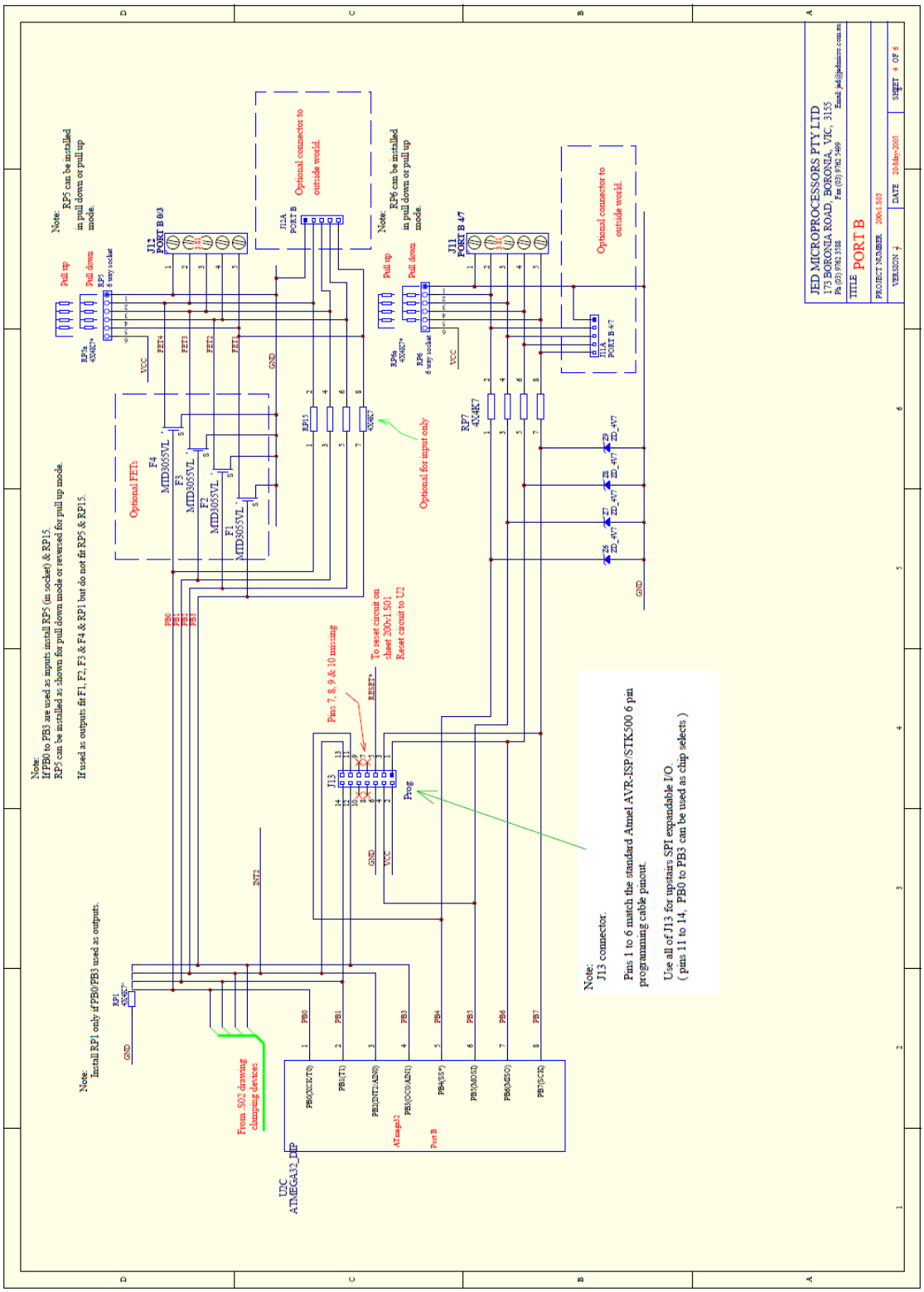
C.3 Circuit Schematics Port A of JED AVR200

Circuit diagram pertaining to Section 8.1.2 (including Subsections 8.1.2.1 to 8.1.2.6)



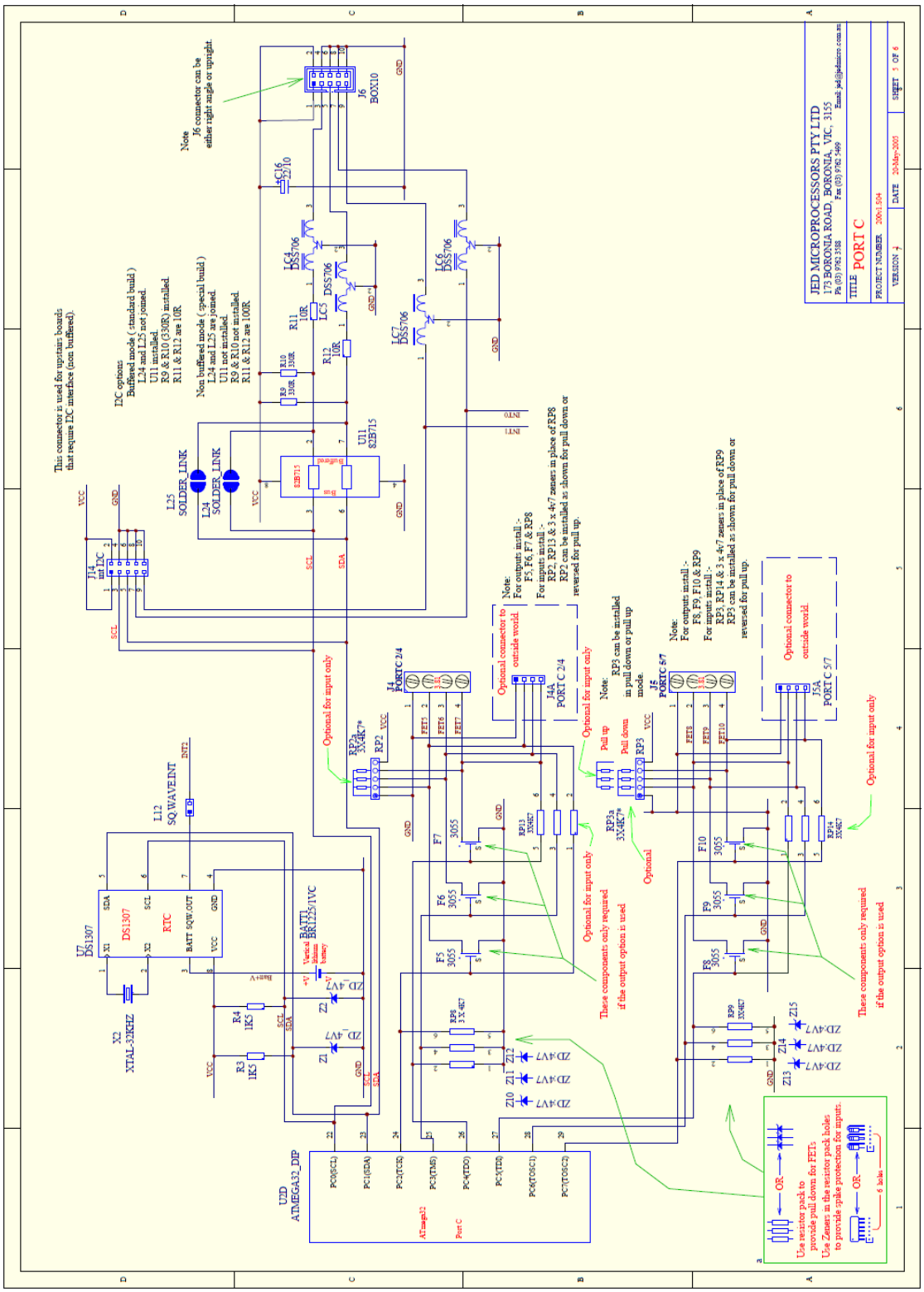
C.4 Circuit Schematics Port B of JED AVR200

Circuit diagram pertaining to Section 8.1.2 (including Subsections 8.1.2.1 to 8.1.2.6)



C.5 Circuit Schematics Port C of JED AVR200

Circuit diagram pertaining to Section 8.1.2 (including Subsections 8.1.2.1 to 8.1.2.6)



C.6 Circuit Schematics Port D of JED AVR200

Circuit diagram pertaining to Section 8.1.2 (including Subsections 8.1.2.1 to 8.1.2.6)